Computers & Operations Research 71 (2016) 71–81

Contents lists available at ScienceDirect

Computers & Operations Research

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

A filter-and-fan approach with adaptive neighborhood switching for resource-constrained project scheduling

Jieguang He [a,b], Xindu Chen [a,*], Xin Chen [a]

[a] Key laboratory of Computer Integrated Manufacturing System. Guangdong University of Technology, Guangzhou, Guangdong, People's Republic of China
[b] Department of Computer Science and Technology, Guangdong University of Petrochemical Technology, Maoming, Guangdong, People's Republic of China

ARTICLE INFO

Available online 21 January 2016

Keywords:
Resource-constrained project scheduling
Local search
Filter-and-fan
Multiple neighborhoods
Adaptive neighborhood switching

ABSTRACT

This paper presents an effective heuristic algorithm based on the framework of the filter-and-fan (F&F) procedure for solving the resource-constrained project scheduling problem (RCPSP). The proposed solution methodology, called the filter-and-fan approach with adaptive neighborhood switching (FFANS), operates on four different neighborhood structures and incorporates improved local search, F&F search with multiple neighborhoods and an adaptive neighborhood switching procedure. The improved local search, in which a new insert-based move strategy and new time compression measurement of schedules having the same makespan are embedded, is utilized to identify a local optimum and a basic move list. The F&F search, aimed to further improve the local optimum, applies multi-neighborhood filter and fan strategies to generate compound moves and a neighborhood-switch list in a tree search fashion. When the current neighborhood cannot further improve the local optimum, the adaptive neighborhood switching procedure picks the most potential neighborhood for the next run of the local search procedure. The entire solution procedure is autonomous and adaptive due to its variable search range depending on the project sizes and characteristics. Computational results and comparisons with some state-of-the-art algorithms indicate the effectiveness and competence of the proposed FFANS.

1. Introduction

The resource-constrained project scheduling problem (RCPSP), which is known to be strongly NP-hard [1], is one of the most intractable combinatorial optimization problems in scheduling [2]. Some exact algorithms [3–5] and a variety of heuristic approaches have emerged to solve the problem in recent decades. However, since the RCPSP is NP-hard, these exact algorithms cannot find the optimal solution within a reasonable time if the scale of a project is relatively large with more than 60 activities or higher resource constraints [6]. Therefore, heuristic procedures remain the only feasible way to solve large and highly resource-constrained projects. Many researchers have made substantial efforts in the development of heuristics to tackle this type of problems in acceptable time. We classify these methodologies into the following categories: priority rule-based X-pass methods [7–10], classical metaheuristics [11–19], population-based methods [20–23], hybrid metaheuristics [24–28], and local search-oriented methods [29–32]. For more details on other heuristics and their comparisons, the readers can refer to the surveys [33–36].

Based on the above-mentioned references, we can see that population-based algorithms as well as hybrid methodologies appear to be able to produce the best results when applied to the RCPSP. But these powerful algorithms generally incorporate one or more intricate guiding strategies, e.g., genetic algorithm (GA), scatter search (SS), particle swarm optimization (PSO), or ant colony optimization (ACO), into the iterative process, thereby resulting in more parameter settings and highly complex parameter adjustments, using, e.g., the trial-and-error strategy, or Taguchi method of design-of-experiment (DOE) [23]. At the same time, there are not many local search-oriented methods for the RCPSP, and usually they cannot perform as well as the above approaches. Besides, almost all of the stopping criteria of the heuristics are determined by a predefined maximal number of constructed schedules or time limit. Although it may be a requirement of an experimental test, a heuristic method is hardly able to autonomously determine the search level (when to stop) in line with the problem scale and the current state of the search without the considerations of predefined criteria.

The filter-and-fan (F&F) approach proposed by Glover [37], which also belongs to the local search-based methodology, has been successfully applied to several NP-hard combinatorial optimization problems, such as the uncapacitated facility (or warehouse) location problem (UFLP) [38], job shop scheduling problem (JSSP) [39], and 2D HP model of the protein folding problem [40]. Hence, we believe that

* Corresponding author.
E-mail addresses: hubice@163.com (J. He), chenxindu@gdut.edu.cn (X. Chen), chenx@gdut.edu.cn (X. Chen).

http://dx.doi.org/10.1016/j.cor.2016.01.012
0305-0548/

the F&F method is a very promising approach to solve the RCPSP. However, to the best of our knowledge, there is only one research by Ranjbar [32] that deals with the RCPSP based on the F&F method. In his paper, the author uses a basic F&F framework, comprising of local and F&F search, for solving the RCPSP. Also, the search procedure is terminated using a predefined time limit.

Consequently, we aim to develop a competitive local search-oriented method that can achieve a high level of overall performance being effective, efficient and robust, and completely adaptive and autonomous. Our proposed algorithm is also based on the framework of the F&F procedure. However, distinguished from the original F&F for the RCPSP [32], named as the O-FF herein, we develop new filter and fan strategies by using multiple neighborhoods in the tree search procedure, and add an adaptive neighborhood switching procedure to the new algorithm. In this paper, a new filter-and-fan approach with adaptive neighborhood switching (FFANS) is presented.

The remainder of this paper is organized as follows. In Section 2, the RCPSP is described briefly. Section 3 introduces some basic definitions and techniques that will be used in the proposed method. Section 4 presents the FFANS algorithm. Computational results, analysis and comparison are provided in Section 5. Finally, we end this paper with some conclusions and directions for further research in Section 6.

2. Problem statement

The RCPSP can be stated as follows. A set of activities N, numbered from 0 to $n+1$ where the activities 0 and $n+1$ are dummy activities indicating the start and end of a project, is to be scheduled without preemption on a set R of renewable resources. An activity p $(p \in N)$ has to be processed for d_p time units with the occupation of r_{pk} units of resource k $(k \in R)$ in each period of its execution. The capacity of resource k is constant throughout the project horizon and limited to R_k, therefore, the sum of resource k consumption at any time period t, $R_k(t)$, cannot exceed R_k. Each activity p has a set of immediate predecessors P_p and a set of immediate successors S_p. Let AP_p be the set of all predecessors of activity p and AS_p be the set of all its successors. The real activities, indexed form 1 to n, have positive duration and nonnegative resource use, while the dummy activities are assumed to not require any resources or processing time. The objective of the RCPSP is to minimize the project makespan. Fig. 1 illustrates an example of a project in the form of an activity-on-node (AON) network with 12 activities, including two dummy activities $p=0$ and 11.

3. Preliminaries

In this section, we briefly describe some definitions and techniques used to build the FFANS.

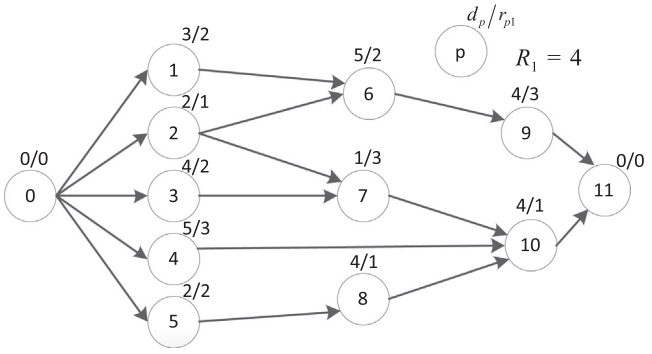

Fig. 1. An AON network as a RCPSP example.

3.1. Forward and backward precedence-feasible activity lists

In this research, the representations of all solutions are in the form of a precedence-feasible activity list, consisting of forward and backward lists. A forward precedence-feasible activity list can be described as $l_{FPF}=(a_0, a_1, \ldots, a_i, \ldots, a_j, \ldots, a_{n+1})$ where for $i<j$, $a_i \in P_{a_j}$, and i, j denote order indexes in the list, a_i represents the ith activity. Similarly, $l_{BPF}=(a_0, a_1, \ldots, a_i, \ldots, a_j, \ldots, a_{n+1})$ where for $i<j$, $a_i \in S_{a_j}$.

3.2. Serial and parallel schedule generation schemes with forward or backward scheduling

Generally, there are two kinds of schedule generation schemes (SGSs) used for decoding a certain activity list, either the serial SGS (SSGS) or the parallel SGS (PSGS). The main difference between these scheduling procedures is that the SSGS is based on activity increase, scheduling one activity at a time, but the PSGS is based on time increase, scheduling possibly more than one activity at a time [41]. Besides, contrary to the SSGS, the PSGS is sometimes unable to reach an optimal solution; nevertheless, it can often acquire a better schedule in a relatively short span of time.

Furthermore, by applying the SSGS or the PSGS on l_{FPF}, which is actually a forward scheduling, the forward SSGS (F-SSGS) or forward PSGS (F-PSGS) can be constructed. Similarly, the backward SSGS (B-SSGS) and the backward PSGS (B-PSGS) can perform a backward scheduling by utilizing l_{BPF}. We consider all of these SGSs for the multi-neighborhood searching in our study.

3.3. Forward and backward schedules

A forward schedule S_F is generated after executing some kind of forward SGS. At the same time, a backward schedule S_B is derived from a backward SGS. The S_F and S_B are defined as follows:

$S_F=((s_{a_0}, f_{a_0}), (s_{a_1}, f_{a_1}), \ldots, (s_{a_i}, f_{a_i}), \ldots, (s_{a_j}, f_{a_j}), \ldots, (s_{a_{n+1}}, f_{a_{n+1}}))$ subject to $s_{a_i}+d_{a_i} \le s_{a_j}$, where $a_i \in P_{a_j}$ and $R_k(t) \le R_k$ for all k, t, and s_{a_i} and f_{a_i} are the start and finish time of activity a_i, respectively.

$S_B=((f_{a_0}, s_{a_0}), (f_{a_1}, s_{a_1}), \ldots, (f_{a_i}, s_{a_i}), \ldots, (f_{a_j}, s_{a_j}), \ldots, (f_{a_{n+1}}, s_{a_{n+1}}))$ subject to $f_{a_i}-d_{a_i} \ge f_{a_j}$, where $a_i \in S_{a_j}$ and $R_k(t) \le R_k$ for all k, t, and s_{a_i} and f_{a_i} are the start and finish time of activity a_i, respectively.

According to different SGS approaches, S_F can be subdivided into F-SSGS-based schedule and F-PSGS-based schedule, namely, S_{F-SSGS} and S_{F-PSGS}. Similarly, S_B is subdivided into S_{B-SSGS} and S_{B-PSGS}.

3.4. Forward and backward standard ordering and forward and backward standard activity lists

For any l_{FPF}, by combing with its S_F, which can be generated by the F-SSGS or the F-PSGS, the employment of forward standard ordering (FSO) can transform it into a forward standard activity list $l_{FS}=(a'_0, a'_1, \ldots, a'_i, \ldots, a'_j, \ldots, a'_{n+1})$, where for $i<j$, $s_{a'_i} < s_{a'_j}$, or $s_{a'_i} = s_{a'_j}$ and $a'_i < a'_j$. In other words, the FSO first sorts l_{FPF} in the ascending order of start times, and then sorts the activities having same start times using their numbers. The backward standard ordering (BSO) first sorts l_{FPF} in the descending order of finish times, and then sorts the activities having same finish times using their numbers, generating a backward standard activity list $l_{BS}=(a'_0, a'_1, \ldots, a'_i, \ldots, a'_j, \ldots, a'_{n+1})$, where for $i<j$, $f_{a'_i} > f_{a'_j}$, or $f_{a'_i} = f_{a'_j}$ and $a'_i > a'_j$.

In a similar way, for l_{BPF} with its derived S_B, obtained by either the B-SSGS or the B-PSGS, l_{FS} and l_{BS} can be built by the FSO and the BSO, respectively.

3.5. Serial and parallel schedule generation schemes with standard forward or backward scheduling

Combining with the above-mentioned standard ordering, four different SGSs can generate four corresponding *standard SGSs*, namely the SF-SSGS, SF-PSGS, SB-SSGS and SB-PSGS. Since they have almost the same steps, we take the SF-SSGS as an example for illustration, which is shown in Algorithm 1.

Algorithm 1. Procedure of the SF-SSGS.

1: **Input:** S
2: **Output:** l_{FS}
3: $l_{FS} \leftarrow$ FSO(S)
4: **If** schedule S is not generated by F-SSGS **then**
5: $S_F \leftarrow$ F-SSGS(l_{FS})
6: $l_{FS} \leftarrow$ FSO(S_F)
7: **End if**
8: **Return** l_{FS}

3.6. Forward–backward improvement

The forward–backward improvement (FBI) or double justification, which was first introduced by Li and Willis [42], and then explained and extended by Valls et al. [17], is an effective method to reduce the makespan of a project by iteratively utilizing the forward and backward SSGS scheduling until no gaps in the schedule can be compressed further. With the help of the SF-SSGS and SB-SSGS, the FBI can be easily incorporated into the FFANS to refine the solution quality. Accordingly, for S_F, the FBI first employs the SB-SSGS and then the SF-SSGS to improve the makespan, and for S_B, the order is reversed.

4. Filter-and-fan approach with adaptive neighborhood switching for RCPSP

In this section, we propose a new filter-and-fan approach with adaptive neighborhood switching (FFANS) for solving the RCPSP. We first depict an insert-based activity move and four different neighborhood structures that are employed in the multi-neighborhood search. Then we illustrate the fundamental components of the proposed algorithm in detail. Finally, we explain how these components work together to fulfill the multi-neighborhood search and their adaptability.

4.1. Activity move

Fleszar and Hindi [29] have developed an enhanced move strategy by taking account of all predecessors and successors, whether direct or indirect, of an activity. According to the strategy, with regard to l_{FPF}, each activity a_i can move to the left (right) position i' as far as its left (right) limit, namely, $FLL(a_i) \le i' \le FRL(a_i)$, where $FLL(a_i) = |AP_{a_i}|$ and $FRL(a_i) = n - |AS_{a_i}| - 1$, where i starts from 0. However, in their study, moving an activity a_i from position i to i' has to progressively exchange adjacent activities that are located on this path. Even worse, if an activity whose position is exactly where a_i may be moved to has a precedence relationship with a_i, it will be forced to move to its left (right). Hence, this method may cause a lot of swaps among the activities. In order to enhance the efficiency and applicability, an insert-based move strategy has been proposed in this paper, which is executed by inserting a list of activities without precedence relations with a_i after (before) a_i. The main steps of this procedure, the forward activity list move (FALM), for l_{FPF} are shown in Algorithm 2, and an example based on Fig. 1 is illustrated in Fig. 2. The activity 7 marked

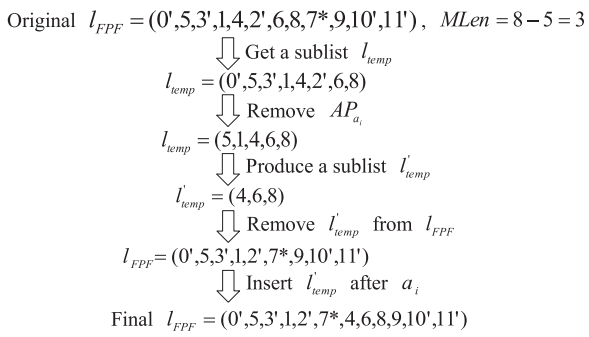

Fig. 2. An example of an insert-based move.

with * is the one that will be moved to position 5, and its predecessors and successors, both direct and indirect, are marked with ′.

Algorithm 2. Procedure of the FALM.

1: **Input:** l_{FPF}, a_i, $toPos$
2: **Output:** l_{FPF}
3: Calculate the move length of activity a_i, $MLen \leftarrow i - toPos$
4: **If** $MLen > 0$ **then**
5: Get a sublist l_{temp} from l_{FPF}, whose activity indexes are from 0 to $i-1$ in l_{FPF}
6: Remove all the predecessors AP_{a_i} of a_i from l_{temp}
7: Produce a sublist l'_{temp} by getting the last $MLen$ activities from l_{temp}
8: Remove l_{temp}' from l_{FPF}
9: Insert l'_{temp} after a_i
10: **Else**
11: Get a sublist l_{temp} from l_{FPF}, whose activity indexes are form $i+1$ to $n+1$ in l_{FPF}
12: Remove all the successors AS_{a_i} of a_i from l_{temp}
13: Produce a sublist l'_{temp} by getting the first $|MLen|$ activities from l_{temp}
14: Remove l'_{temp} from l_{FPF}
15: Insert l'_{temp} before a_i
16: **End if**
17: **Return** l_{FPF}

On the other side, for l_{BPF}, the backward activity list move (BALM) can be constructed similarly. It should be pointed out that the left and right limits of activity a_i belonging to l_{BPF} are defined as $BLL(a_i) = |AS_{a_i}|$ and $BRL(a_i) = n - |AP_{a_i}| - 1$, respectively.

Compared to the simple move strategy [35], the insert-based move strategy proposed here can obtain a greater range of possible moves, because the range is unacted on the current sequence of activities. An activity can be moved to any legal position, which is related to the number of its predecessors or successors, but not the current position of its immediate predecessor or successor. But using the simple move strategy, the activity can be moved just to the position after its immediate predecessor or before its immediate successor as far as possible. Moreover, compared to the enhanced move strategy [29] based on activity swaps, the proposed method is more efficient and universal, in the sense that regardless of whether there is a predecessor or successor in the position to which the activity is moved, the algorithm is carried out in the same way; but using the enhanced move method, if there is a predecessor or successor in that position, the predecessor or successor must be moved to the left or right first for making room for the moving activity, thus causing a lot of swaps among the activities. Hence, the insert-based move strategy is able to generate any feasible list.

4.2. Neighborhood structures

By integrating the FALM with the F-SSGS and the F-PSGS, respectively, two neighborhood structures (*NS*) can be developed. And by integrating the BALM with the B-SSGS and the B-PSGS, respectively, we can generate two other neighborhood structures. The F-SSGS, F-PSGS, B-SSGS and B-PSGS also represent four different neighborhood structures.

4.3. Filter-and-fan procedure with adaptive neighborhood switching

4.3.1. Obtaining an initial schedule

In the proposed algorithm, the roulette selection method is adopted to generate an initial schedule. During the process of constructing the schedule, the latest finish time (LFT) priority rule [43] is applied to generate the probabilities to decide which activity will be selected from the eligible set formed by the former activity. At the same time, a certain SGS is used to calculate the start or finish time of the selected activity.

4.3.2. Improved local search

In the process of local search, especially in its later phase, plenty of different schedules possessing the same value of the makespan are generated. Therefore, for the sake of choosing some more promising schedules among those having the same makespan, a new indicator time compression (*TC*) is defined here.

For a forward schedule S_F, the *TC* is calculated by

$$TC = \sum_{i=0}^{n+1} s_i \tag{1}$$

And for a backward schedule S_B, the *TC* is calculated by

$$TC = \sum_{i=0}^{n+1} horizon - f_i \tag{2}$$

where the horizon is the upper bound of the makespan, which can be obtained as the sum of the durations of the activities. The use of the *TC* measure primarily lies on the following observations. On the one hand, if some schedules have the same value of the makespan, then the one with the smallest *TC* can usually be relatively easily improved in the later process of moving; on the other hand, a schedule having a smaller *TC* might be more easily compressed by the FBI procedure. The main reason of the above phenomena is that it is generally easier for the activities that appear near the end of the standard list with smaller *TC* to find available resources.

In this study, we develop an improved local search algorithm by incorporating the *TC* measurement of a schedule into the restricted neighborhood search proposed by Fleszar and Hindi [29]. At the same time, the improved local search can realize four different search patterns owing to corresponding neighborhood structures. The improved local search is shown in the pseudo-code in Algorithm 3.

Algorithm 3. Local search.

Input: S, NS
Output: S, M
Get the corresponding list l from the schedule S
Divide l into I equal parts, $l_1, \ldots, l_i, \ldots, l_I$, such that each part has α activities and the last part is the remainder
For $i = 1$ **to** I **do**
  Create an empty schedule list SL
  For each a_p $(p = 0, \ldots, \alpha - 1)$ in the l_i **do**
    If NS=F-SSGS or NS=F-PSGS **then**
      For $k = FLL(a_p)$ **to** $FRL(a_p)$ **do**
        Make a new list $l_{FPF} \leftarrow \mathrm{FALM}(l, a_p, k)$
        Decode l_{FPF} according to the NS, and obtain a new schedule S_{new}
      End for
    Else
      For $k = BLL(a_p)$ **to** $BRL(a_p)$ **do**
        Make a new list $l_{BPF} \leftarrow \mathrm{BALM}(l, a_p, k)$
        Decode l_{BPF} according to the NS, and obtain a new schedule S_{new}
      End for
    End if
    Put S_{new} into SL
  End for
  Get the best schedule S_{best} from SL, whose makespan is the smallest or *TC* is the smallest if there are more than one schedule having the same smallest makespan
  If S_{best} is better than S **then** // the makespan of S_{best} is smaller than that of S
    $S \leftarrow S_{best}$
    Restart the Local Search
  End if
End for
Return (S, M); // M is a basic-move list (see Section 4.3.3)

It is worth noting that the improved local search has its own characteristics besides those of the restricted neighborhood search. These characteristics can be summed up as follows:

(1) The procedure is actually the descent search using a candidate list strategy. All the legal moves concerning α activities form a candidate list at each stage i. In addition, the indicator *TC* can help to choose the most potential candidate (solution) as the new starting point, if some solutions have the same makespan.
(2) The circular search fashion of the activities and multiple neighborhoods enable the local search to increase the diversity of the solutions. This is because when the search progresses, the composition of candidate lists changes; moreover, different solutions can be produced in different neighborhoods, resulting in different schedules by corresponding SGSs. In fact, in different neighborhoods, namely, F-SSGS and F-PSGS, or B-SSGS and B-PSGS, even the same solution can generate different schedules because of different SGSs.
(3) The procedure is also similar to the Tabu search. When a best move is found for α activities leading to a reduction in the makespan, the search restarts from where it reaches. So, it is usually impossible to move an activity back.

4.3.3. Generating a list of basic moves

During the local search, a list M of basic moves is generated, which is one of the most important parts of the F&F procedure. The list M contains η_0 moves associated with the best schedules found by the local search, so that the best legitimate trial moves used to fan new schedules can be selected from it in the process of the tree search. In this research, a simple but effective method is proposed to select moves for constructing the list M. A move is denoted as a triple $m = (j, p, q)$, meaning that activity j at position p can move to position q, and each move has two evaluation indexes, the makespan and the *TC*, which are obtained from the corresponding schedule. Each activity usually possesses multiple positions where it can be moved to, especially in some large-scale RCPSPs having many parallel activities. Therefore, only those most powerful moves, whose selection criteria are equivalent to those of S_{best} s, will be stored in M. On the other side, in order to fan diversified solutions at the stage of the tree search, we choose λ different best moves for each activity. The value of λ is set to satisfy the Eq. (3), whose detailed explanation is presented in Section

4.3.4.

$$\lambda n \geq 2 * \eta_1 \eta_2 \tag{3}$$

Furthermore, if the number of the potential positions is less than λ, the best move is replicated so that the number of possible moves is λ.

4.3.4. Filter-and-fan search with multiple neighborhoods

Given any schedule S associated with the *NS*, the local search iteratively improves S using the *NS* until a local optimum is found, and finally produces a list M of $\eta_0 = \lambda n$ component moves. The current local optimum is then used as the root for constructing an F&F tree with L levels. In this study, for each level, we design a *fan candidate list strategy with multiple neighborhoods* to fan a higher number of trial solutions containing some generated by other neighborhoods, and a *filter candidate list strategy with multiple neighborhoods* to select the solutions generated by the current neighborhood for the next level and store all the solutions of the other neighborhoods in the *neighborhood-switch list*.

The process of the F&F search is actually that of constructing an F&F tree, which is based on alternate use of the above strategies for successive levels. First, an enhanced-move is defined here to distinguish from the basic move stored in the list M, which is composed of two different basic moves. Denote the enhanced-move as $em = ((i, p_i, q_i), (j, p_j, q_j))$. The reason for using an enhanced-move is that it contributes to further enlarging of the search scope with almost no increase in the time consumption. So, even for the first level, an enhanced-move may generate a new solution better than the root. And empirical results show that the employment of two basic moves as an enhanced-move can produce a relatively perfect exploring range, instead of prematurely ignoring local optimal solutions.

Next, in the first level, based on the current *NS*, η_1 strictly different enhanced-moves defining the candidate list $EM(0)$ are used to create the nodes of the F&F tree being individually applied to the root. Strict differences among enhanced-moves imply that there are no identical basic moves in $EM(0)$. At the same time, according to the other three *NS*s, the root generates other three solutions using the corresponding standard SGSs. If the best solution generated by the current *NS* is better than the root, the F&F search stops immediately and goes back to the local search restarting with this newly improved solution. It should be noted that the solutions generated by the other *NS* s are not engaged in the above selection but are stored in the neighborhood-switch list. If no improved solution is found, the search continues constructing the next level. The next level is created as follows. Let η_1 be the number of enhanced-moves in $EM(k)$. The method proceeds by selecting a set of η_2 strict enhanced-moves from M associated with each solution $S_i(k)$ $(i = 1, \ldots, \eta_1)$ to generate $\eta = \eta_1 * \eta_2$ trial solutions for level $k+1$ (as a result of applying η_2 enhanced-moves to each solution of level k) with the current *NS*. As indicated in [39], setting $\eta_1 = 2 * \eta_2$ has been found to yield a good tradeoff between the time complexity and performance of the search. Likewise, by applying the standard SGSs corresponding to the other three *NS*s to each solution $S_i(k)$ $(i = 1, \ldots, \eta_1)$, $3 * \eta_1$ solutions of the other neighborhoods can be generated and then put into the neighborhood-switch list. If an improving enhanced-move is found within the trial solutions, the search stops branching and turns back to the descent local search with the improved solution. If not, a filter procedure is employed to select the η_1 "best" enhanced-moves from the set of η enhanced-moves generated for the new $EM(k)$. The term "best" does not necessarily means that the schedules created by the best enhanced-moves should have the smallest makespans or TC s, but indicates that the best schedules need to abide by the diversification criterion. The diversification criterion can be obtained by making the distance between each pair of η_1 selected schedules not less than a threshold value th_dist. In this study, we measure the distance between two schedules based on the position of each activity in a standard activity list, which can be calculated as follows:

$$dist(l_{S_1}, l_{S_2}) = \frac{1}{n+2} \sum_{i=0}^{n+1} |\text{position of activity } i \text{ in } l_{S_i} - \text{position of activity } i \text{ in } l_{S_2}| \tag{4}$$

where l_{S_1} and l_{S_2} are forward or backward standard activity lists. Afterwards, the next level $k+1$ can be built by applying the $EM(k)$ enhanced-moves to the corresponding solutions $S_i(k)$ to create $S_i(k+1)$. At last, if the solution at the root node cannot be improved after reaching the maximum number of levels L, the F&F search returns to the neighborhood switching procedure with the last best solution obtained at the level L. Moreover, whenever the search goes back to the neighborhood switching procedure, the neighborhood-switch list is sent back too. The outline of our F&F search algorithm with multiple neighborhoods (FFSMN) is shown in Algorithm 4.

Algorithm 4. The filter-and-fan search with multiple neighborhoods.

Input: S_{root}, *NS*, M //S_{root} is the root node of the search tree
Output: $S_{currentNS}$, *NSL*
$k \leftarrow 1$
Create a candidate list $EM(0)$ with η_1 strict enhanced-moves from M
  // the basic moves comprising the enhanced-moves can be chosen randomly
Apply the $EM(0)$ enhanced-moves to S_{root} to create the first level of the F&F tree with solutions $S_i(k)$ $(i = 1, \ldots, \eta_1)$
Generate three solutions for S_{root} by applying the standard SGSs corresponding to the other three *NS*s, and store all of these solutions in a neighborhood-switch list *NSL*
Find the best solution $S_{currentNS}$ from $S_i(k)$ $(i = 1, \ldots, \eta_1)$
If $S_{currentNS}$ is better than S_{root} **then**
  Return ($S_{currentNS}$,*NSL*);
Else
  Do
    Identify η_2 strict enhanced-moves derived from M for each solution $S_i(k)$ $(i = 1, \ldots, \eta_1)$, and evaluate each one of them by computing the value of the corresponding trial solution
    Generate three solutions for each solution $S_i(k)$ $(i = 1, \ldots, \eta_1)$ by applying the standard SGSs corresponding to the other three *NS*s, and store all of them in *NSL*
    Find the best solution $S_{currentNS}$ from the $\eta = \eta_1 * \eta_2$ trial solutions
    If $S_{currentNS}$ is better than S_{root} **then**
      Return ($S_{currentNS}$,*NSL*)
    Else
      Select the best η_1 trial moves, whose corresponding trial solutions must abide by the diversification criterion, to become the members of $EM(k)$
      Apply the $EM(k)$ enhanced-moves to the corresponding solutions $S_i(k)$ to create $S_i(k+1)$
    End if
    $k \leftarrow k+1$
  While $k \leq L$
End if
Return ($S_{currentNS}$, *NSL*)

In brief, to build up an F&F tree, two stages, the *fanning stage* and *filtering stage*, need to be carried out alternately. For the level k with η_1 solutions, the level $k+1$ can be constructed as follows:

(1) The fanning stage. According to the current *NS*, each solution generates η_2 new solutions by using η_2 enhanced-moves, each comprising of two different basic moves, and then these new

solutions are evaluated by the current SGS. Moreover, each solution also generates three other neighborhoods' solutions by applying the corresponding standard SGSs.

(2) The filtering stage. For the $\eta=\eta_1*\eta_2$ solutions generated in the fanning stage, the best η_1 ones are chosen to construct the level $k+1$ according to the diversification criterion.

To illustrate the process of the F&F search consider an instance j12019_2.sm encompassing 120 activities and 4 renewable resources, which is a member of the J120 dataset obtained from the well-known library PSPLIB designed by Kolish and Sprecher [44]. Because the F&F tree of this instance is very large, we just display a segment of it, in which some key nodes of the first three levels and a true trajectory to find the best solution $S_{currentNS}$ have been reserved. So for this reduced example, as shown in Fig. 3, the parameters can be set as: $\eta_1=3$, $\eta_2=2$, $L=3$, $th\_dist=2$. Nodes in the tree represent solutions with the corresponding object function values, each of which includes its makespan and *TC*. There are two types of arcs in the tree, namely, the other-neighborhood-fan arc denoting use of the other neighborhood structures, and the current-neighborhood-fan arc denoting use of the current neighborhood structure.

The current *NS* is B-PSGS, and the entry solution is obtained from the local search using the BALM and the B-PSGS decoding method. For level 1, the entry solution generates three new solutions in the current *NS*, namely, solutions A, B and C. Using solution A as an example, the enhanced-move ((47,77,61),(110,25,4)) is operated on the entry solution to obtain a new list, then the B-PSGS is applied to decode the list. Finally solution A is evaluated as the makespan 87 and *TC* 5094. Meanwhile, the entry solution also generates three other neighborhoods' solutions, namely, solutions 1, 2 and 3. Solution 1 is produced by the SF-SSGS, so the entry solution of the B-PSGS neighborhood has been changed to the one of the F-SSGS neighborhood. Solutions 2 and 3 are carried out in a similar way. Solutions A, B and C are all selected as the members of level 1, and solutions 1, 2 and 3 are stored in a neighborhood-switch list. For level 2, the generation of new solutions is the same as for level 1, but each solution generates two new solutions, so the filter strategy depicted above is utilized to select the best ones. The solution E has the same makespan as the solution G, but a smaller *TC*. So solution E is chosen. The reason for abandoning solutions F, G and I is that they disobey the criterion of the diversification. For level 3, the best solution $S_{currentNS}$ has been found, and then the F&F search returns to the local search with this new starting point. It is worth noting that in the level 2 an improved solution $S_{otherNS}$ has appeared by using the F-PSGS neighborhood structure, but the F&F search does not check the evaluations of this solution and just put it into the neighborhood-switch list. The search only returns to the local search when an improved solution generated by the current neighborhood has been found, for example, $S_{currentNS}$ found in the level 3. Hence, switching from a neighborhood to another one happens only after the current neighborhood has been explored as much as possible.

4.3.5. *The general filter-and-fan procedure with adaptive neighborhood switching*

The adaptive neighborhood switching procedure is the most important part of our proposed F&F method, which can effectively extend search neighborhoods to further overcome local optimality. After returning from the F&F search, neighborhood switching is executed if the current neighborhood cannot improve the local optimum anymore. The neighborhood switching strategies mainly comprise: (1) the lock strategy which is used to lock the current neighborhood, (2) the unlock strategy which is used to unlock all neighborhoods or just a certain neighborhood, (3) the alternative neighborhood selection strategy to identify the next legitimate neighborhood and a new starting solution for another run of the local search procedure, and (4) the termination criterion. The general design of our filter-and-fan procedure with adaptive neighborhood switching is depicted in Algorithm 5. All the four neighborhoods are unlocked before the process starts. For the sake of clearness, we define S^*_{local} as the local best solution obtained by the local search, S^*_{tree} as the best solution obtained by the F&F search, $S^*_{otherNS}$ as the best solution obtained from the other neighborhoods, and S^*_{so-far} as the best solution found so far. Clearly, S^*_{local} is the value passed to S_{root} in the FFSMN, and S^*_{tree} is the returned value from the FFSMN, which is equal to $S_{currentNS}$. The FFANS is described in Algorithm 5.

Algorithm 5. The general filter-and-fan procedure with adaptive neighborhood switching for the RCPSP.

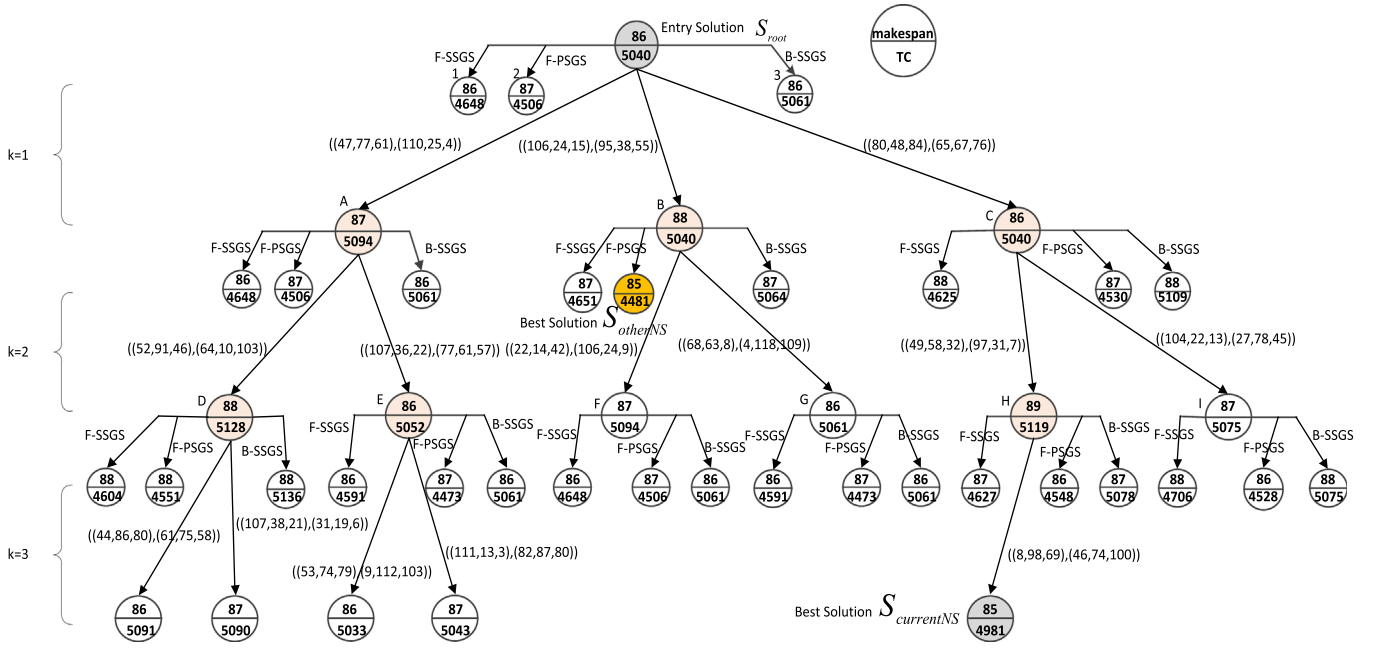

Fig. 3. Example of a filter-and-fan tree with multiple neighborhoods for j12019_2.sm ($\eta_1=3$, $\eta_2=2$, $L=3$, $th\_dist=2$).

Input: S, NS
// S is an initial schedule, and NS is an initial neighborhood structure, which can be any one of the F-SSGS, F-PSGS, B-SSGS and B-SSGS.
$S^*_{so-far} \leftarrow S$
Create an empty general neighborhood-switch list *GNSL*
Do
  $(S^*_{local}, M) \leftarrow$ LocalSearch(S,NS)
  If S^*_{local} is better than S^*_{so-far} **then**
    Unlock all neighborhoods
  End if
  $(S^*_{tree}, NSL) \leftarrow$ FFSMN(S^*_{local},NS,M)
  Put *NSL* into *GNSL*
  If S^*_{tree} is better than S^*_{so-far} **then**
    $S^*_{so-far} \leftarrow S^*_{tree}$
    $S \leftarrow S^*_{tree}$
    Unlock all neighborhoods
  Else if S^*_{tree} is better than S^*_{local} **then**
    $S \leftarrow S^*_{tree}$
  Else
    Lock current *NS*
    Get $S^*_{otherNS}$ from *GNSL* // regardless of whether the neighborhood corresponding to $S^*_{otherNS}$ has been locked or not
    If $S^*_{otherNS}$ is better than S^*_{so-far} **then**
      $S^*_{so-far} \leftarrow S^*_{otherNS}$
      $S \leftarrow S^*_{otherNS}$
      Unlock all neighborhoods
      $NS \leftarrow S^*_{otherNS}.NS$
    Else if $S^*_{otherNS}$ is better than S^*_{local} **then**
      $S \leftarrow S^*_{otherNS}$
      $NS \leftarrow S^*_{otherNS}.NS$
      Unlock *NS*
    Else
      Get the first best solution $S_{otherNS}$ from *GNSL*, whose neighborhood is not locked
      $S \leftarrow S_{otherNS}$
      $NS \leftarrow S_{otherNS}.NS$
    End if
    Clear *GNSL*
  End if
While there is at least one unlocked neighborhood

The general procedure is dynamic and adaptive because all of its components, the local search, the F&F search and the neighborhood switching procedure, are dynamic and adaptive. These traits can be described as: (1) the basic moves used to make up the list M are chosen based on the local search; (2) the enhanced-moves selected to create the next level and the level at which the F&F search can stop depend on the current state of the F&F search; (3) the components of the neighborhood-switch list are identified by the current neighborhood search that may include several iterations of the local search and F&F search, and the selection of the best solution generated by another neighborhood and its corresponding neighborhood type are determined by the current state of the general search; (4) locking or unlocking a particular neighborhood, or even unlocking of all the neighborhoods, is also done according to the current state of the general search. In summary, the search is always capable of moving towards the most promising neighborhood.

5. Computational experiments

The proposed FFANS algorithm was implemented in Java 1.6 and complied using Eclipse v.4.2. All the experiments were conducted on a PC with AMD Athlon II X4 640 3.0 Gz CPU and 4 GB RAM running Windows 7 Ultimate. The well-known benchmark instance sets from the PSPLB (http://www.om-db.wi.tum.de/psplib/datasm.html) were used to evaluate the algorithm. These datasets can be divided into four parts, namely, J30, J60, J90, and J120, with project sizes of 30, 60, 90, and 120 activities, respectively. Each of the first three sets contains 480 problem instances, and the last set contains 600 problem instances. However for each instance, there are four different types of renewable resources that can be used. Furthermore, the levels of three independent problem parameters, the network complexity (*NC*), resource factor (*RF*), and resource strength (*RS*), are systematically altered to produce each set. For J30, J60, and J90, the levels of the parameters are set as follows: $NC \in \{1.5, 1.8, 2.1\}$, $RF \in \{0.25, 0.5, 0.75, 1\}$, and $RS \in \{0.2, 0.5, 0.7, 1\}$, and for J120, $RS \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$.

5.1. Parameter settings

The setting of parameters is very easy in the FFANS, where appropriate values are obtained by means of empirical fine tuning. The final settings are listed in Table 1. α is set to an empirical value available in the literature [29]. The settings of η_1, η_2 and L have been found to yield a good tradeoff between the number of schedules generated (also reflecting time consumption) and performance of the search for large scale projects, such as J90 and J120. The sets J30 and J60, the smaller projects, also use the same parameter settings of η_1, η_2 and L, because we want to verify the adaptability and autonomy of the algorithm. In fact, the algorithm itself is able to adapt search strength to the project size and characteristics (*NC*, *RF*, and *RS*) even under the same parameter settings. It is important to note that the value of λ does not need to be adjusted, and is determined by Eq. (3).

5.2. Computational results

The assessment of the FFANS is embodied in the solution quality, and the solution quality is commonly measured by the average relative percentage deviation (Dev.) from the critical-path based lower bound for J60, J90, and J120; and for J30, the lower bound is based on the optimal solution. Moreover, the indexes of the average/max number of schedules generated (Av.No.sch./MaxNo.sch.) and average/max CPU time (Av.CPU/MaxCPU) are applied to measure the efficiency of the proposed approach.

Table 2 lists the results obtained by the FFANS, where four different initial *NS* s have been tested for each set.

5.2.1. Overviews

From Table 2, it can be seen that no matter which *NS* the algorithm starts from, it can eventually provide roughly the same solution quality, with the largest gap between different *NS*s being in the case of the B-SSGS and B-PSGS where the difference in Dev. of J120 reaches 0.07%. But for a more accurate identification, if the problem scale is small, the SSGSs (forward or backward) are better choices for the initial *NS*s, for example, for the J30 and J60 sets. However, for a moderate or larger scale, the PSGSs (forward

Table 1
Parameter settings in the FFANS.

Problem set	λ	α	η_1	η_2	L	*th_dist*
J30	20	7	24	12	20	2
J60	10					
J90	7					
J120	5					

or backward) show advantages in solution quality and computational time. The larger the project scale, the better the PSGSs as the initial *NS*s; for example, for the J90 and J120 sets. The reasons can be explained as follows. The search space of the project whose size is big or *RS* is low is usually very huge, but the utilization of PSGSs can make the search be confined to a subspace of it, namely, the set of non-delay schedules; therefore the solutions generated by the current *NS* (F-PSGS or B-PSGS) as well as the other *NS*s, are better than those generated by SSGSs during the primary stage of the search, and this advantage increases the probability of continuing searching. However, for smaller size projects or the ones with a higher *RS*, the search space is much smaller, and PSGSs may eliminate all optimal solutions from the search space, so SSGSs are better choices. The above results also confirm the findings of Kolisch [41] and Hartmann and Kolisch [34]. Furthermore, another thing worth mentioning is that the use of backward scheduling SGSs (B-SSGS or B-PSGS) has advantages over forward scheduling SGSs (F-SSGS or F-PSGS) in the beginning of the search, except the set J90. (The difference in Dev. produced by the F-PSGS and B-PSGS is 0.01% only, so the backward scheduling is still competitive.)

5.2.2. Adaptability analysis

From the microscopic perspective, the adaptive property of the FFANS mainly manifests in locking and unlocking of one or more neighborhoods according to the current state of the search. But on the macro level, the algorithm is capable to adjust its search degree, characterized by the number of schedules generated after the search is done, to the problem scale, *RS* of the project, and initial *NS*. We analyze the adaptability of the algorithm from these three aspects.

(1) Problem scale. The problem scale is the most significant factor contributing to Av.No.sch. as well as Av.CPU. As the size of the problem increases, the average number of generated schedules and average CPU time increase monotonically in a nonlinear way. The result means that in order to keep high quality of the solutions, the algorithm autonomously changes the size of the search range in line with the problem scale.

(2) Resource strength. *RS* is another important factor that is related to Av.No.sch. and Av.CPU, which can be seen in Figs. 4 and 5 that are obtained from the set J90 and use the F-PSGS as the initial *NS*. As we know, there are 48 different combinations, each comprising 10 instances, in the set J90, and these combinations can be further divided into 12 groups on the basis of the values of *RS* ($RS \in \{0.2,\ 0.5,\ 0.7,\ 1\}$). So we figure out Av.No.sch.-10 (Av.CPU-10) by averaging the number of the generated schedules (CPU time) of every 10 instances. In each group, both Av.No.sch.-10 and Av.CPU-10 decrease monotonically with increase in *RS*. These results also indicate that the algorithm can adapt to changes in *RS* to find better solutions.

(3) Initial neighborhood structure. Based on further investigation, we know that, although different initial *NS*s have different advantages when applied to different sizes of projects, the average deviations, finally obtained, are almost the same. The reason is that the procedures that are dominated in the beginning of the search exert more effort to examine more schedules for producing consistently high quality solutions. For example, the value of Av.No.sch. associated with the F-SSGS or B-SSGS is larger than that of the F-PSGS or F-PSGS in the set J120; however, the situation is reversed for the set J30. Hence, the above analysis demonstrates that the proposed FFANS is an adaptive method.

Table 2
Overview of the results for the FFANS.

Problem set	Indexes	Initial *NS*s			
		F-SSGS	F-PSGS	B-SSGS	B-PSGS
J30	Dev. (%)	0.00	0.00	0.00	0.00
	Av.No.sch.	96,710	99,009	96,722	99,870
	MaxNo.sch.	128,422	163,441	146,765	191,865
	Av.CPU (s)	2.83	2.88	2.82	2.87
	MaxCPU (s)	3.75	4.74	4.45	5.63
J60	Dev. (%)	10.52	10.54	**10.51**	10.52
	Av.No.sch.	134,174	138,014	137,372	138,544
	MaxNo.sch.	351,667	355,860	535,356	457,641
	Av.CPU (s)	8.05	8.07	8.11	8.04
	MaxCPU (s)	22.19	20.54	30.95	26.23
J90	Dev. (%)	9.86	**9.85**	9.88	9.86
	Av.No.sch.	205,718	210,456	206,924	206,362
	MaxNo.sch.	729,161	809,800	741,150	715,280
	Av.CPU (s)	24.10	24.79	24.93	24.57
	MaxCPU (s)	83.98	121.79	83.54	87.91
J120	Dev. (%)	30.77	30.72	30.75	**30.70**
	Av.No.sch.	472,692	454,423	485,802	466,017
	MaxNo.sch.	2,019,025	1,575,175	2,209,877	1,522,269
	Av.CPU (s)	122.44	110.19	116.30	109.32
	MaxCPU (s)	537.20	426.30	549.70	360.60

The bold values mean they are the best results in different initial NSs.

5.2.3. Robustness analysis

The FFANS possesses remarkable robustness due to its good adaptability. That is to say, the algorithm is robust because it can provide high quality solutions over a wide range of problems that have considerably different characteristics and sizes. For further testifying this, we use the set J90 as an example to compare our best solutions (OB) with the so-far-best ones (SFB). Using similar grouping and calculating methods as those proposed above, we draw two broken lines, Dev.-10_OB and Dev.-10_SFB, in Fig. 6. As can be seen in the figure, the line Dev.-10_OB coincides almost exactly with the line Dev.-10_SFB. The biggest difference between them is only 2.81% for group 13 (averaging from j9013_1 to j9013_10).

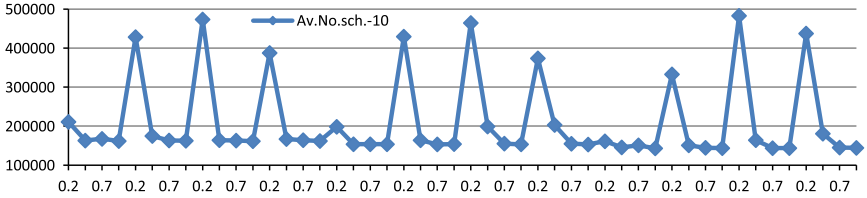

Fig. 4. Effect of *RS* on the average number of generated schedules (set J90).

5.3. Comparison with other algorithms

In this subsection, we will compare the FFANS with existing state-of-the-art algorithms based on the PSPLIB datasets to further demonstrate its effectiveness. This comparison is composed of two parts. First, we compare the FFANS with local search methodologies, because it is also a local search-oriented method. And then, the comparison proceeds to other metaheuristics, including famous GAs, population based approaches and hybrids. However, unlike other metaheuristics, local search-based algorithms generally generate quite a number of schedules. In addition to the adaptability of the FFANS, it is possible to restrict the FFANS to generating 1000 or 5000 schedules only. In the present context, only the local search procedure of the FFANS may produce more than 5000 schedules. Under any circumstances, it is not that increasing the number of schedules that a particular heuristic is allowed to produce would help to discover better solutions, since that depends on the neighborhood search strategy which can effectively extend the search scope. Therefore, in the following comparisons, we usually only list those most powerful algorithms that are used to examine 50,000 or even more schedules in the literature.

Table 3 summarizes the results of the comparison of the FFANS with other local search approaches. The values in parentheses denote the number of schedules generated or time consumed. But for the FFANS, the number of schedules, CPU time and the best initial *NS* are all recorded. It is rather clear that there are just few studies that utilize local search methods in a single solution framework, and the FFANS outperforms all such local search based methods. Moreover, compared to the O-FF that is also based on the framework of the F&F approach, the multi-neighborhood filter and fan strategies and the adaptive neighborhood switching procedure make the FFANS more effective. And comparing to the classic local search based method VNS, we find not only that our FFANS method is better in solution quality, but that it also produces much less schedules.

As shown in Table 4, the FFANS can find all the optimal solutions in the set J30. Regarding the set J60, the FFANS is the second best with $>50{,}000$ schedules, and the gap between the FFANS

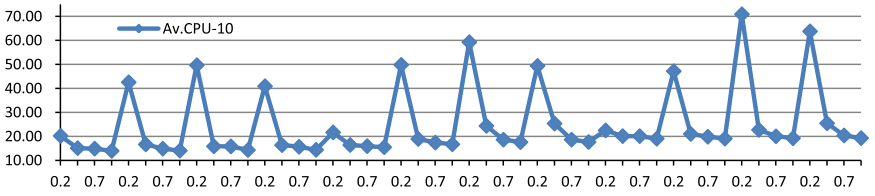

Fig. 5. Effect of *RS* on the average CPU time (set J90).

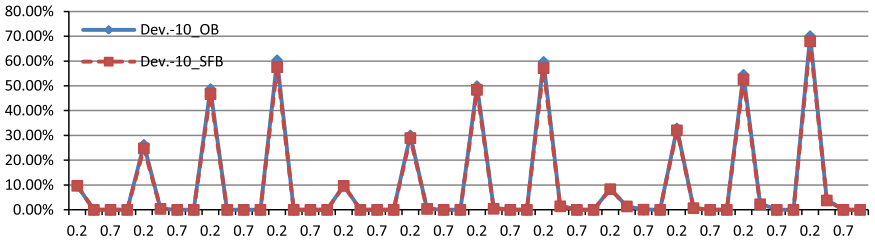

Fig. 6. Comparison of our best solutions with the so-far-best ones by Dev.-10 (set J90).

Table 3
Comparison of local search metaheuristics based on the PSPLIB.

Problem set	Algorithm	Reference	Dev. (%)		
			5000	50,000	>50,000
J30	FFANS	This paper	–	–	0.00(96,722, B-SSGS, 2.82 s)
	AILS	Paraskevopoulos et al. [45]	0.01	0.00	–
	O-FF	Ranjbar [32]	–	–	0.00(5 s)
	LSSPER	Palpant et al. [30]	0.00	–	–
	VNS	Fleszar and Hindi [29]	–	–	0.01(0.64 s)
J60	FFANS	This paper	–	–	10.51(137,372, B-SSGS, 8.11 s)
	O-FF	Ranjbar [32]	–	–	10.56(5 s)
	LSSPER	Palpant et al. [30]	10.81	–	–
	AILS	Paraskevopoulos et al. [45]	11.10	10.91	–
	VNS	Fleszar and Hindi [29]			10.94(152,503)
J90	FFANS	This paper	–	–	9.85(210,456, F-PSGS, 24.79 s)
	O-FF	Ranjbar [32]	–	–	10.11(5 s)
J120	FFANS	This paper	–	–	30.70(466,017,B-PSGS, 109.32 s)
	O-FF	Ranjbar [32]	–	–	31.42(5 s)
	LSSPER	Palpant et al. [30]	32.41	–	–
	AILS	Paraskevopoulos et al. [45]	32.67	32.66	–
	VNS	Fleszar and Hindi [29]	–	–	33.10(1,874,641)

and the best algorithm is only 0.05%. There are very few existing test results for the set J90, so, we consider the two current best results to compare with our algorithm. The FFANS shows good performance, being in the second place. As for the set J120, the FFANS is ranked the fifth, and the gap between the FFANS and the best algorithm is just 0.31%. Moreover, although the number of schedules generated by our solution procedure is quite large, the average computational time is acceptable. Hence, the FFANS is an effective and competitive algorithm for solving the RCPSP.

6. Conclusions and further research

In this paper, we developed a new local search-based method for solving the well-known resource constrained project scheduling problem. The proposed method, called the FFANS, relies on the framework of the filter-and-fan method and is composed of improved local search, filter-and-fan search with multiple neighborhoods and an adaptive neighborhood switching procedure. First of all, we design four different neighborhood structures by simply combining two SGSs with two directions of scheduling. At the local search stage, an insert-based move strategy is used to efficiently shift the activities, and new measurement time compression is proposed to select most potential ones from the schedules having the same makespan. At the filter-and-fan search stage, the multi-neighborhood filter and fan strategies are employed to construct a filter-and-fan tree as well as a neighborhood-switch list. Further, the enhanced-move is used as a replacement for the basic move for fanning more effective trial solutions. As for the neighborhood switching procedure, it can single out the most promising neighborhood and a new starting solution for the next run of the local search procedure if the current neighborhood fails to improve the local optimum. The entire solution procedure is completely autonomous and adaptive, because it is able to autonomously switch to another neighborhood according to the current state of the search and decide the search range (strength) depending on the problem scale and characteristics. Computational experiments based on the PSPLIB demonstrate that the FFANS can produce consistently high quality solutions regardless of the project sizes. And comparison to the existing most powerful algorithms also shows that the proposed method is effective and competitive. In summary, the FFANS is able to make a significant contribution to the field of local search-based algorithms.

The most promising avenue for future research is to design a more effective move selection mechanism for enhancing the filter and fan strategies. On the other hand, in the filter-and-fan search stage, dynamically updating the basic move list according to the current state of the search may further enhance the diversity of trial solutions. Furthermore, a big step somersaulting strategy,

Table 4
Comparison to other competitive metaheuristics based on the PSPLIB.

Problem set	Algorithm	Reference	Dev. (%)		
			5000	50,000	> 50,000
J30	FFANS	This paper	–	–	0.00(96,722, B-SSGS, 2.82 s)
	SAILS	Paraskevopoulos et al. [45]	0.01	0.00	–
	GA, TS-PR	Kochetov and Stolyar [46]	0.04	0,00	–
	GA-MBX	Zaman [15]	0.04	0.00	–
	SS-PR	Mahdi Mobini et al. [47]	0.02	0.01	–
	ACOSS	Chen et al. [27]	0.06	0.01	
	GAPS	Mendes et al. [14]	0.02	0.01	–
	SS-FBI	Debels et al. [26]	0.11	0.01	0.01(500,000)
	DBGA	Debels and Vanhoucke [13]	0.04	0.02	–
	GA-hybrid FBI	Valls et al. [12]	0.06	0.02	–
	SFLA	Fang and Wang [23]	0.21	0.18	–
J60	SAILS	Paraskevopoulos et al. [45]	10.72	10.54	10.46(70,000)
	FFANS	This paper	–	–	10.51(137,372, B-SSGS, 8.11 s)
	SS-FBI	Debels et al. [26]	11.10	10.71	10.53(500,000)
	GA-MBX	Zamani [15]	10.94	10.65	10.54(150,000)
	SS-PR	Mahdi Mobini et al. [47]	10.74	10.57	–
	SFLA	Fang and Wang [23]	10.87	10.66	–
	ACOSS	Chen et al. [27]	10.98	10.67	–
	GAPS	Mendes et al. [14]	11.04	10.67	10.67(63,546)
	DBGA	Debels and Vanhoucke [13]	10.95	10.68	–
	GA-hybrid FBI	Valls et al. [12]	11.10	10.73	–
	GA, TS-PR	Kochetov and Stolyar [46]	11.17	10.74	–
J90	SS-FBI	Debels et al. [26]	10.59	10.09	9.80(500,000)
	FFANS	This paper	–	–	9.85(210,456, F-PSGS, 24.79 s)
	DBGA	Debels and Vanhoucke [13]	10.95	10.68	–
J120	SAILS	Paraskevopoulos et al. [45]	32.12	30.78	30.39(200,000)
	SS-FBI	Debels et al. [26]	33.10	31.57	30.48(500,000)
	ACOSS	Chen et al. [27]	32.48	30.56	–
	DBGA	Debels and Vanhoucke [13]	32.18	30.69	–
	FFANS	This paper	–	–	30.70(466,017,B-PSGS, 109.32 s)
	GA-MBX	Zamani [15]	32.48	31.30	30.75(150,000)
	SFLA	Fang and Wang [23]	33.20	31.11	–
	GAPS	Mendes et al. [14]	33.03	31.44	31.20(127,341)
	GA-hybrid FBI	Valls et al. [12]	32.54	31.24	–
	SS-PR	Mahdi Mobini et al. [47]	32.61	31.37	–
	GA, TS-PR	Kochetov and Stolyar [46]	33.36	32.06	–

which is used to find a new starting solution far away from all the current found best solutions when all neighborhoods have been locked, is a direction worth pursuing to overcome local optimality.

Acknowledgments

This work is supported by the National Science & Technology Pillar Program of China (No. 2012BAF12B10).

References

[1] Blazewicz J, Lenstra JK, Rinnoy K. Scheduling projects subject to resource constraints: classification and complexity. Discrete Appl Math 1983;5(1):11–24.
[2] Möhring RH, Schulz AS, Stork F, Uetz M. Solving project scheduling problems by minimum cut computations. Manag Sci 2003;49(3):330–50.
[3] Klein R, Scholl A. Scattered branch and bound: an adaptative search strategy applied to resource-constrained project scheduling problem (Technical report). Darmstadt: University of Technology; 1998.
[4] Mingozzi A, Maniezzo V, Ricciardelli S, Bianco L. An exact algorithm for project scheduling with resource constraints based on a new mathematical formulation. Manag Sci 1998;44(5):714–29.
[5] Sprecher A. Scheduling resource-constrained projects competitively at modest memory requirements. Manag Sci 2000;46(5):710–23.
[6] Alcaraz J, Maroto C. A robust genetic algorithm for resource allocation in project scheduling. Ann Oper Res 2001;102(1–4):83–109.
[7] Kolisch R. Efficient priority rules for the resource-constrained project scheduling problem. J Oper Manag 1996;14(3):179–92.
[8] Özdamar L, Ulusoy G. An iterative local constraint based analysis for solving the resource constrained project scheduling problem. J Oper Manag 1996;14(3):193–208.
[9] Boctor FF. Some efficient multi-heuristic procedures for resource-constrained project scheduling. Eur J Oper Res 1990;49(1):3–13.
[10] Ulusoy G, Özdamar L. Heuristic performance and network/resource characteristics in resource-constrained project scheduling. J Oper Res Soc 1989;40(12):1145–52.
[11] Hartmann S. A self-adapting genetic algorithm for project scheduling under resource constraints. Naval Res Logist 2002;49(5):433–48.
[12] Valls V, Ballestin F, Quintanilla S. A hybrid genetic algorithm for the resource-constrained project scheduling problem. Eur J Oper Res 2008;185(2):495–508.
[13] Debels D, Vanhoucke M. A decomposition-based genetic algorithm for the resource-constrained project-scheduling problem. Oper Res 2007;55(3):457–69.
[14] Mendes JJM, Goncalves JF, Resende MGC. A random key based genetic algorithm for the resource constrained project scheduling problem. Comput Oper Res 2009;36(1):92–109.
[15] Zamani R. A competitive magnet-based genetic algorithm for solving the resource-constrained project scheduling problem. Eur J Oper Res 2013;229(2):552–9.
[16] Bouleimen K, Lecocq H. A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version. Eur J Oper Res 2003;149(2):268–81.
[17] Valls V, Ballestín F, Quintanilla S. Justification and RCPSP: a technique that pays. Eur J Oper Res 2005;165(2):375–86.
[18] Artigues C, Michelon P, Reusser S. Insertion techniques for static and dynamic resource-constrained project scheduling. Eur J Oper Res 2003;149(2):249–67.
[19] Pan NH, Hsaio PW, Chen KY. A study of project scheduling optimization using Tabu Search algorithm. Eng Appl Artif Intell 2008;21(7):1101–12.
[20] Merkle D, Middendorf M, Schmeck H. Ant colony optimization for resource-constrained project scheduling. IEEE Trans Evol Comput 2002;6(4):333–46.
[21] Zhang H, Li X, Li H, et al. Particle swarm optimization-based schemes for resource-constrained project scheduling. Automat Constr 2005;14(3):393–404.
[22] Ziarati K, Akbari R, Zeighami V. On the performance of bee algorithms for resource-constrained project scheduling problem. Appl Soft Comput 2011;11(4):3720–33.
[23] Fang C, Wang L. An effective shuffled frog-leaping algorithm for resource-constrained project scheduling problem. Comput Oper Res 2012;39(5):890–901.
[24] Valls V, Quintanilla S, Ballestín F. Resource-constrained project scheduling: a critical activity reordering heuristic. Eur J Oper Res 2003;149(2):282–301.
[25] Tseng LY, Chen SC. A hybrid metaheuristic for the resource-constrained project scheduling problem. Eur J Oper Res 2006;175(2):707–21.
[26] Debels D, De Reyck B, Leus R, Vanhoucke M. A hybrid scatter search/electromagnetism meta-heuristic for project scheduling. Eur J Oper Res 2006;169(2):638–53.
[27] Chen W, Shi Y, Teng H, Hu L. An efficient hybrid algorithm for resource-constrained project scheduling. Inf Sci 2010;180(6):1031–9.
[28] Agarwal A, Colak S, Erenguc S. A neurogenetic approach for the resource-constrained project scheduling problem. Comput Oper Res 2011;38(1):44–50.
[29] Fleszar K, Hindi KS. Solving the resource-constrained project scheduling problem by a variable neighbourhood search. Eur J Oper Res 2004;155(2):402–13.
[30] Palpant M, Artigues C, Michelon P. LSSPER: solving the resource-constrained project scheduling problem with large neighbourhood search. Ann Oper Res 2004;131(1–4):237–57.
[31] Valls V, Quintanilla S, Ballestín F. Resource-constrained project scheduling: a critical activity reordering heuristic. Eur J Oper Res 2003;149(2):282–301.
[32] Ranjbar M. Solving the resource-constrained project scheduling problem using filter-and-fan approach. Appl Math Comput 2008;201(1):313–8.
[33] Kolisch R, Hartmann S. Experimental investigation of heuristics for resource-constrained project scheduling: An update. Eur J Oper Res 2006;174(1):23–37.
[34] Hartmann S, Kolisch R. Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem. Eur J Oper Res 2000;127(2):394–407.
[35] Kolisch R, Hartmann S. Heuristic algorithms for solving the resource-constrained project scheduling problem: classification, computational, analysis. In: Weglarz J, editor. Project scheduling: recent models, algorithms and applications. Berlin: Kluwer Press; 1999. p. 147–78.
[36] Kolisch R, Padman R. An integrated survey of deterministic project scheduling. Omega 2001;29(3):249–72.
[37] Glover F. A template for scatter search and path relinking. In: Hao JK, Lutton E, Ronald E, Schoenauer M, Snyers D, editors. Artificial evolution, lecture notes in computer science. Heidelberg: Springer Press; 1998. p. 13–54.
[38] Greistorfer P, Rego C. A simple filter-and-fan approach to the facility location problem. Comput Oper Res 2006;33(9):2590–601.
[39] Rego C, Duarte R. A filter-and-fan approach to the job shop scheduling problem. Eur J Oper Res 2009;194(3):650–62.
[40] Rego C, Li H, Glover F. A filter-and-fan approach to the 2D HP model of the protein folding problem. Ann Oper Res 2011;188(1):389–414.
[41] Kolisch R. Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. Eur J Oper Res 1996;90(2):320–33.
[42] Li KY, Willis RJ. An iterative scheduling technique for resource-constrained project scheduling. Eur J Oper Res 1992;56(3):370–9.
[43] Davis EW, Patterson JH. A comparison of heuristic and optimum solutions in resource-constrained project scheduling. Manag Sci 1975;21(7):944–55.
[44] Kolisch R, Sprecher A. PSPLIB—a project scheduling problem library: OR software-ORSEP operations research software exchange program. Eur J Oper Res 1997;96(1):205–16.
[45] Paraskevopoulos DC, Tarantilis CD, Ioannou G. Solving project scheduling problems with resource constraints via an event list-based evolutionary algorithm. Expert Syst Appl 2012;39(4):3983–94.
[46] Kochetov Y, Stolyar A. Evolutionary local search with variable neighborhood for the resource constrained project scheduling problem. In: Proceedings of the 3rd international workshop of computer science and information technologies, Russia; 2003.
[47] Mahdi Mobini MD, Rabbani M, Amalnik MS, Razmi J, Rahimi-Vahed AR. Using an enhanced scatter search algorithm for a resource-constrained project scheduling problem. Soft Comput 2009;13(6):597–610.