CIRRELT

Centre interuniversitaire de recherche
sur les réseaux d'entreprise, la logistique et le transport

Interuniversity Research Centre
on Enterprise Networks, Logistics and Transportation

A Path Relinking-Based Scatter Search for the Resource-Constrained Project Scheduling Problem

François Berthaut
Robert Pellerin
Adnène Hajji
Nathalie Perrier

October 2014

CIRRELT-2014-50

Document de travail également publié par la Faculté des sciences de l'administration de l'Université Laval, sous le numéro FSA-2014-008.

Bureaux de Montréal :
Université de Montréal
Pavillon André-Aisenstadt
C.P. 6128, succursale Centre-ville
Montréal (Québec)
Canada H3C 3J7
Téléphone : 514 343-7575
Télécopie : 514 343-7121

Bureaux de Québec :
Université Laval
Pavillon Palasis-Prince
2325, de la Terrasse, bureau 2642
Québec (Québec)
Canada G1V 0A6
Téléphone : 418 656-2073
Télécopie : 418 656-2624

www.cirrelt.ca

A Path Relinking-Based Scatter Search for the Resource-Constrained Project Scheduling Problem

François Berthaut[1,2,*], Robert Pellerin[1,2], Adnène Hajji[1,3], Nathalie Perrier[2]

[1] Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT)

[2] Department of Mathematical and Industrial Engineering, Polytechnique Montréal, C.P. 6079, succursale Centre-ville, Montréal, Canada H3C 3A7

[3] Department of Operations and Decision Systems, 2325 de la Terrasse, Université Laval, Québec, Canada G1V 0A6

Abstract. Project scheduling has received a growing attention from researchers for the last decades in order to propose models and methods to tackle scheduling problems for real-size projects. In this paper, we consider the resource-constrained project scheduling problem (RCPSP), which consists in scheduling the activities in order to minimize the project duration in presence of precedence and resource constraints. As this problem is NP-hard, we propose a hybrid metaheuristic based on scatter search that involves forward-backward improvement and reversing the project network at each iteration of the scatter search. A bidirectional path relinking method with a new move is used as combination method. An advanced parameter tuning method based on local search is employed. The proposed method is applied on the standard benchmark projects of size 30, 60 and 120 activities from the PSPLIB library and compared with the state-of-the-art heuristics of the literature. The computational results show that the proposed hybrid scatter search produces high-quality solutions in reasonable computational time and is among the best performing metaheuristics.

Keywords. Resource-constrained project scheduling, metaheuristics, makespan minimization, scatter search, path relinking.

Acknowledgments. This work has been supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), the Jarislowsky/SNC-Lavalin Research Chair in the Management of International Projects, and the CIRRELT. This support is gratefully acknowledged.

---

* Corresponding author: Francois.Berthaut@cirrelt.ca

Dépôt légal – Bibliothèque et Archives nationales du Québec
Bibliothèque et Archives Canada, 2014

1. Introduction

Project scheduling consists in determining the start and end times of the project activities in presence of scarce resources and precedence relations. Since the early 1960's, this problem has attracted both researchers and practitioners. For the latter, scheduling serves major functions and is often supported in practice by a project planning software which models the project, solves the scheduling problem and provides representations for communication. For researchers, project scheduling is very attractive because the models are rich, have a variety of applications and are difficult to solve (Brucker et al., 1999). The standard project scheduling problem is the resource-constrained project scheduling problem (RCPSP), which involves the determination of a precedence- and resource-feasible schedule that minimizes the project duration. Even though the assumptions of this problem do not cover all the situations that occur in practice, many more general models, variants and extensions have been developed, often using the RCPSP as starting point. For more details on the research on RCPSP, the reader is referred to the following surveys: Özdamar and Ulusoy (1995) Herroelen et al. (1998), Brucker et al. (1999), Kolisch and Hartmann (1999, 2006), Hartmann and Kolisch (2000), Kolisch and Padman (2001), Tavares (2002), Herroelen (2005), Hartmann and Briskorn (2010).

The RCPSP belongs to the class of NP-hard optimization problems, and hence, is one of the most intractable problems in operations research (Blazevicz et al., 1983). Whereas surveys presented in Icmeli-Tukel and Rom (1998), Liberatore et al. (2001), Liberatore and Pollack-Johnson (2003) show that more than half of projects in practice have a size larger than one hundred activities, Herroelen (2005) pointed that exact procedures have only the capability to solve small scale projects with at most sixty activities and slight resource constraints in acceptable computation effort. This limitation has motivated a growing number of academics to develop heuristic procedures able to find good-quality schedules in reasonable computation effort for larger projects usually encountered in practical cases. In their experimental investigation of the best heuristics for the RCPSP, Hartmann and Kolisch (2000) presented the computational results of thirteen heuristics. In the updated version of Kolisch and Hartmann (2006), thirty-seven heuristics were compared. From 2006 to 2013, we identified more than forty additional heuristics for the RCPSP in the literature. The current best performing heuristics are metaheuristics, which are capable of learning. The most efficient of them do not follow classical paradigms but combine concepts of different classical metaheuristics into hybrid approaches and use advanced mechanisms such as the forward-backward improvement (FBI), the path relinking (PR) and specific representations (Herroelen, 2005; Kolisch and Hartmann, 2006).

In the last years, several researchers have proposed hybrid metaheuristics based on the scatter search framework. Conceptualized by Glover (1977), scatter search (SS) is an evolutionary

method inspired from the principle that systematic designs and methods for creating new solutions afford significant benefits beyond those derived from recourse to randomization. The first application of SS for the RCPSP can be found in Valls et al. (2004), who presented a population-based approach that alternates between a forward and backward scheduling procedure to improve the resource utilization and a blend of SS and PR strategies to combine and evolve the population of solutions. Debels et al. (2006) proposed a hybrid SS that incorporates a combination method based on the principles of electromagnetism and uses FBI as improvement method. Ranjbar et al. (2009) used the same SS backbone as in Debels et al. (2006), but reversed the project network at each population generation and combined solution using PR. Mobini et al. (2009) derived an enhanced SS that employs FBI as improvement method and combines solutions with a two point cross-over operator, a PR strategy and a permutation-based operator. Chen et al. (2010) proposed an ant colony optimization that uses SS as improvement method and the resource utilization improvement procedure developed by Valls et al. (2004). Finally, Paraskevopoulos et al. (2012) developed a hybrid SS that combines solutions using a linear combination method based on events' starting times, and improves them with an adaptive iterative local search. Based on the experimental results on the PSPLIB benchmark (Kolisch and Sprecher, 1996), these SS based algorithms have been identified as being among the current best heuristics.

This paper is motivated by the attractive results obtained with SS for the RCPSP and contributes to the long-term objective of developing efficient metaheuristics that can handle large-sized projects and that can be adapted for more complex scheduling problems. We propose a new hybrid metaheuristic that relies on a SS skeleton inspired from Debels et al. (2006), Ranjbar et al. (2009) and Mobini et al. (2009). In particular, this metaheuristic involves FBI and reversing the project network at each iteration. Moreover, the proposed combination method presents two distinctive features. It is based on a bidirectional PR, which has been identified as a promising approach for combination in SS by Glover et al. (1995), Marti et al. (2006) and Resende et al. (2010). This PR generates solutions by exploring a path build by gradual moves between two solutions to be combined. Also, we introduce a new move that consists in moving the most distant activity from the guiding solution in the current solution of the path to its place in the guiding solution. The computational results show that the proposed hybrid scatter search produces high-quality solutions in reasonable computational time and is among the best performing metaheuristics.

The remainder of this paper is organized as follows. Section 2 introduces the definition of the RCPSP and the notations. Section 3 provides a description of the proposed hybrid SS. In Section 4, an advanced parameter tuning method based on local search is proposed and computational results are presented, including a comparison with the state-of-the-art heuristics from the

literature. Finally, we conclude by summarizing the main results and by highlighting possible directions for further research in Section 5.

2. Resource-Constraint Project Scheduling Problem definition

The RCPSP problem can be defined as follows. A single project is composed of a set N of n activities labeled j = 1,...,n, and two dummy activities 0 and n+1, which are the project start and project end, respectively. We denote by d_j the duration of activity j. The activities have to be performed without preemption. Technological reasons imply that some activities must finish before others can start. These precedence relations are denoted by $i \rightarrow j$. P_j and S_j represent the set of immediate predecessors and successors of activity j, respectively. Each activity requires constant amounts of renewable resources to be performed. These resources are called renewable because they are fully available at each period. There are K resource types and we denote by r_{ik} the usage of resource $k \in K$ per period by activity $j \in N$. The constant capacity of each resource type is represented by R_k. The two dummy activities have zero duration and no resource usage. All information is assumed to be deterministic and known in advance. The parameters are assumed to be nonnegative and integer-valued.

A schedule is defined as an assignment of start times s_j and finish times f_j to activities $j \in N$. A schedule is called precedence-feasible if $s_j \geq f_i$ for each precedence constraint $i \rightarrow j$, and resource-feasible if the consumption of resources does not exceed the resource capacity for each time period and each resource type. The RCPSP consists in determining a precedence and resource-feasible schedule so as to minimize the project duration (project makespan).

3. Hybrid Scatter Search Algorithm

Introduced by Glover (1977), SS is an evolutionary algorithm that generates new solutions by combining preserved ones. SS is composed of the following basic methods (Marti et al., 2006):

- diversification generation method: to generate an initial population of trial solutions;
- improvement: to transform a trial solution into enhanced solutions;
- reference set update method: to build and maintain a reference set of the best solutions, that is often partitioned into a set of high-quality solutions and a set of diversified solutions;
- subset generation method: to produce subsets of solutions of the reference set. The common method is to generate pairs of reference solutions;
- solution combination method: to transform a given subset of solutions produced by the subset generation method into one or more combined solutions.

SS is a very flexible methodology since each element can be implemented with different degree of sophistication. It uses strategies for intensification and diversification that have been proved effective in a variety of optimization problems (Marti et al., 2006).

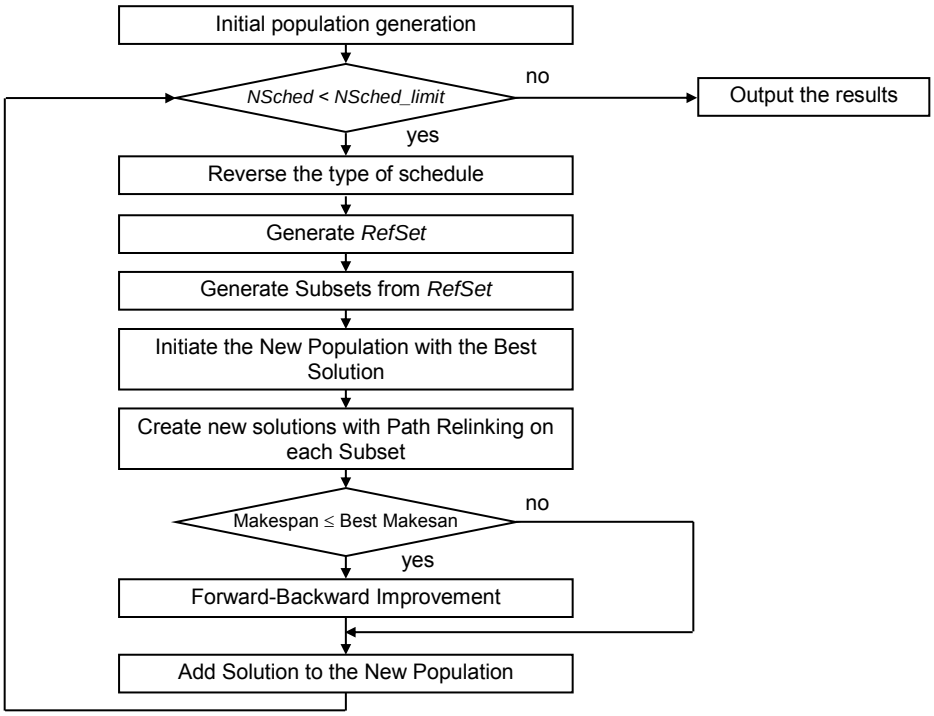

Figure 1 Flow Chart of the proposed Scatter Search algorithm

depicted in Figure 1. An initial population of size *InitPop* is first generated. A reference set *RefSet* composed of two distinct sets of high-quality and diversified solutions is build from the population of solutions. These reference solutions will be evolved to form a new population. This new population is first initialized with the best current solution. The reference solutions are afterwards paired to form subsets that are combined with PR to generate new solutions. These solutions are evaluated and either directly added to the new population or first improved by FBI depending on their quality. The algorithm stops when the number of generated schedules reaches *NSched_limit*. At each iteration, the scheduling direction and the project network are reversed. The solution representation and decoding, and the details of each step are described below.

3.1. Solution representation and decoding

Usually, metaheuristic approaches for the RCPSP operate on representations of schedules rather than directly on schedules themselves (Kolisch and Hartmann, 1999). A schedule generation scheme (SGS), is then required to transform a representation into a feasible schedule. Two different SGSs are usually considered in the literature: the serial and the parallel SGS. The

dominant representations are the activity list and the random key. Experimental investigations have shown that the best metaheuristics employ the serial SGS combined with the activity list representation (Hartmann and Kolisch, 2000; Kolisch and Hartmann, 2006). We consequently choose this combination of methods for the proposed SS.

The serial SGS starts from scratch and builds a feasible schedule by stepwise extension of a partial schedule. At each stage, an activity is selected from the eligible set and scheduled at the earliest precedence- and resource-feasible start time. The eligible set comprises all the activities whose predecessors are already scheduled. A representation of schedules is required to drive the selection of activities. The activity list representation makes use of a sequence of all activities $AL = (a_1,\ldots, a_n)$ in which the position of an activity determines its relative priority with the other activities for being selected. We also define the vector $P = (p_1,..., p_n)$ to represent the position of each activity in AL. If $i = a_h$, the activity i is in position $p_i = h$. As defined, an AL may not be precedence-feasible. In the literature, most authors use precedence-feasible activity lists in which each activity is placed in the list after all of its predecessors. Hartmann and Kolisch (2000) noted that such lists are faster to decode, since the construction of the eligible set and the selection of an activity do not need to be computed during the serial SGS procedure. Indeed, the serial SGS simply selects at each stage j the activity a_j of the precedence-feasible activity list. Consequently, any AL will be considered as precedence-feasible for the rest of the paper.

A major problem with AL representations is that a single schedule can be represented by more than one activity list. This is caused by timing anomalies and activities with identical starting times, as explained in Debels et al. (2006). Consequently, the solution space is filled with useless replications of representations, and thus, the search efficiency is typically reduced (Paraskevopoulos et al., 2012). To tackle these issues, Valls et al. (2003) introduced the topological order (TO) representation, which is an activity list that satisfies: $s_i < s_j$ implies $p_i < p_j$. Ties are broken by the activity index: $s_i = s_j$ and $i < j$ implies $p_i < p_j$. The TO representation can be obtained from an activity list by decoding it through a serial SGS, and then ordering the activities according to the obtained start times to satisfy the conditions of the TO representation. If TO(S) is the TO representation of a schedule S, then S(TO(S)) = S. In other words, this guarantees that any TO representation is uniquely associated with a schedule. We adopt a modified version of the TO representation, explained in the next section.

3.2. Direct and Reverse network

Ranjbar et al. (2009) proposed a SS that alternates at each iteration between direct and reverse project networks in order to explore a different solution space. They define the direct project network as the network in which the arrows represent the original precedence relations. The reverse project network is obtained by reversing the directions of all the arrows such that the end

(start) activity becomes the start (end) activity and each precedence relation $i \rightarrow j$ becomes $j \rightarrow i$. Any feasible schedule for the direct (reverse) network is called a direct (reverse) schedule. As they obtained promising results, the same mechanism is used in the proposed SS.

In order to embed the TO representation into the reversing of the project network, Ranjbar et al. (2009) adapted the TO representation of Valls et al. (2003) to generate reverse (direct) children from direct (reverse) solutions. A direct (reverse) activity list is first decoded with the serial SGS to obtain the finish times of the activities based on the direct (reverse) project network. The activity list is then reordered based on the non-increasing order of their finish times: $f_i > f_j$ implies $p_i < p_j$, while $f_i = f_j$ and $i > j$ implies $p_i < p_j$. The obtained topological order activity list (TOAL) representation is precedence-feasible for the reverse (direct) network.

3.3. Initial population generation

Regret-biased random sampling (RBRS) with the minimum latest finish time (LFT) is used to generate an initial population of size *InitPop* with the direct project network. Previous studies showed the superiority of random sampling over deterministic approaches (Kolisch 1996). Random sampling generates schedules by biasing a priority rule through a random device. Among these methods, RBRS with LFT is one of the most powerful (Kolisch, 1996; Tormos and Lova, 2001; Valls, 2005). Each activity list is iteratively created by randomly selecting an activity of the decision set D_k according to the following probability to be placed at position k:

$$\psi(i) = \frac{(\rho_i + 1)}{\sum_{j \in D_k} (\rho_j + 1)}, \text{ with } \rho_i = \max_{j \in D_k} (LF_j) - LF_i, \ \forall i \in D_k \quad (1)$$

where LF_i denotes the latest finish time of activity i calculated by the Critical Path Method (CPM), and the decision set D_k represents the eligible set at stage k. The serial SGS is then applied to decode the generated activity lists into schedules and to evaluate their makespan.

3.4. Reverse the type of schedule

Once a new population have been generated, the direction of the project network and the associated type of schedule are reversed from direct (reverse) to reverse (direct) at the beginning of a new SS iteration. As each direct (reverse) activity list of the population has been decoded with the serial SGS, the finish times are used to obtain the TO representation that can be used for scheduling with the reverse (direct) network (see section 3.2).

3.5. RefSet Update

The SS algorithm builds and maintains a subset *RefSet* of high-quality and diverse solutions found during the search. According to the SS terminology proposed in Marti et al. (2006), we use

the so-called static Refset Update mechanism. It is applied on the new population during the process once all the subsets have been combined. The reference set *RefSet* is divided into two disjoint subsets $RefSet_1$ and $RefSet_2$ of size b_1 and b_2, respectively. Each solution of $RefSet_1$ should have a distance of at least t_1 with other solutions of $RefSet_1$, while each solution of $RefSet_2$ should have a distance of at least t_2 (with $t_2 > t_1$) with any solution of *RefSet1* or *RefSet2*. The distance measure between two solutions represented by AL_1 and AL_2 is defined as followed:

$$\text{Distance}(AL_1, AL_2) = \frac{1}{n} \cdot \sum_{i=1}^{i=n} |p_{i1} - p_{i2}| \tag{2}$$

where p_{i1} and p_{i2} are the position of activity i in the activity lists AL_1 and AL_2, respectively. The construction of $RefSet_1$ and $RefSet_2$ starts by sorting the new population *New_Pop* according to the lowest makespan. The following function is considered in case of tie:

$$f(s) = \sum_{i=1}^{i=n} \left(d_i \cdot \frac{f_i - EF_i}{EF_i + 1} \cdot \sum_{k=1}^{K} r_{ik} \right) \text{ for direct schedules,}$$
$$f(s) = \sum_{i=1}^{i=n} \left(d_i \cdot \frac{f_i - (LS_n - LS_i)}{LS_n - LS_i + 1} \cdot \sum_{k=1}^{K} r_{ik} \right) \text{ for reverse schedules} \tag{3}$$

Introduced by Paraskevopoulos et al. (2012), this function depicts for each activity i the deviation of the finish time (f_i) from the earliest finish time (EF_i) and the latest start time (LS_i) calculated by the CPM. This expression is weighted with the activity duration and the sum of the resource consumption. Assuming that high quality solutions have the least deviations from the earliest finish times (latest finish times) in direct (reverse) schedules, it can be expected that the lower this function, the better the quality of the solution.

$RefSet_1$ is initialized with the best solution. The next best solutions of *New_Pop* are scanned and added to $RefSet_1$ if the minimal distance with the current members of $RefSet_1$ is superior or equal to t_1. In a second phase, an improvement algorithm is applied to check if any solutions of *New_Pop* \ $RefSet_1$ can be added or can replace a member of $RefSet_1$ to increase the size of $RefSet_1$ (if inferior to b_1), or to improve the quality (average makespan) or the diversity (average distance with the other members of the set) of $RefSet_1$. $RefSet_2$ is similarly built from the remaining solutions *New_Pop* \ $RefSet_1$. Depending on the values of b_1, b_2, t_1 and t_2, there may not be enough qualified solutions to complete $RefSet_1$ and $RefSet_2$. If this situation arises when the RefSet Update mechanism is applied on the initial population, we complete the refsets with the best remaining solutions without checking the threshold conditions. Otherwise, a diversification strategy involving a frequency-based memory is used to complete the refsets with new diversified activity lists. We maintain for this purpose a matrix M during the search, where each element m_{ij} tracks the number of time activity i has been placed directly before activity j in all

previously generated solutions. A new activity list is built step by step by adding at each stage *k* the activity *j* of the decision set D_k that has the least been placed directly after the activity a_{k-1} at position (*k*-1) in the list under construction: $m_{a_{k-1}\,j} = \min_{r \in D_k} \left(m_{a_{k-1}\,r} \right)$.

3.6. Generate the subsets from Refsets

The subsets generation block forms the subsets of solutions that will be used as basis for creating the combined solutions. In its typical form, the combination method of a SS consists in combining pairs of elements of $RefSet_1$ and $RefSet_2$ (Marti et al., 2006). This method is applied to generate all pairs of solutions in $RefSet_1$, and all combination of one solution from $RefSet_1$ and another one from $RefSet_2$. Therefore, the total number of pairs to be combined is $\frac{b_1 \cdot (b_1 - 1)}{2} + b_1 \cdot b_2$.

3.7. Combination method - Path Relinking

PR is used as combination method to generate the next new population from the subsets of solutions. This evolutionary method was originally designed in the context of tabu search to integrate intensification and diversification strategies, and was later suggested as combination method for SS (Marti et al., 2006; Resende et al., 2010). PR generates new solutions by exploring trajectories that connect high-quality solutions, starting from one solution (the initiating solution) and building a path in the neighborhood space that leads towards other solutions (the guiding solutions). This is accomplished by gradual moves that introduce attributes contained in the guiding solutions.

In our SS algorithm, PR is applied on each pair, which constitutes the initiating and guiding solutions. The moves are performed on the activity list representation of the solutions, AL_1 and AL_2. The proposed PR is a bidirectional PR that swaps the initiating and guiding solutions at each step of the path construction. This advanced type of PR was identified as a promising approach by Glover et al. (1995), Marti et al. (2006) and Resende et al. (2010). Second, we propose a new move that consists in moving the most distant activity from the guiding solution in the current solution of the path to its place in the guiding solution. We initialize the PR by setting the initiating and guiding solutions with $AL_{guid} = AL_1$, and $AL_{init} = AL_2$, such that the makespan of AL_1 is better than the makespan of AL_2. We consider $AL_{guid} = (a_{1,guid}, \ldots, a_{n,guid})$ and $AL_{init} = (a_{1,init}, \ldots, a_{n,init})$, where $a_{i,guid}$ and $a_{i,init}$ represent the activity at position *i* in the activity lists AL_{guid} and AL_{init}, respectively. We denote by *PR_set* the set of activity lists that will be created during the PR process. We introduce the following distance measure for each activity:

$$\text{Distance}(j) = \left| p_{j,init} - p_{j,guid} \right|, \quad \forall j \in [1,.., n] \tag{4}$$

where $p_{j,init}$ and $p_{j,guid}$ represent the position of activity j in the initiating and guiding activity lists, respectively. Let q be the most distant activity between AL_{init} and AL_{guid}, p_{init} be the position of q in AL_{init} ($a_{p_{init},init} = q$), and p_{guid} be the position of q in AL_{guid} ($a_{p_{guid},guid} = q$). From AL_{init}, the current activity list AL_{cu} is built by moving activity q at position p_{guid} as follows:

1) If $p_{init} < p_{guid}$, then

$$AL_{cu} = \left(a_{1,init},..., a_{(p_{init}-1),init}, a_{(p_{init}+1),init},..., a_{(p_{guid}-1),init}, q, a_{p_{guid},init}, a_{(p_{guid}+1),init},..., a_{n,init} \right);$$

2) if $p_{init} > p_{guid}$, then

$$AL_{cu} = \left(a_{1,init},..., a_{(p_{guid}-1),init}, q, a_{p_{guid},init}, a_{(p_{guid}+1),init},..., a_{(p_{init}-1),init}, a_{(p_{init}+1),init},..., a_{n,init} \right).$$

If the current activity list is not precedence-feasible, the following repair mechanism is used:

1) If $p_{init} < p_{guid}$, then for direct (reverse) schedules all successors (predecessors) of q between positions p_{init} and p_{guid} - 1 in AL_{cu} are moved right after q in the same order;
2) If $p_{init} > p_{guid}$, then for direct (reverse) schedules all predecessors (successors) of q between position p_{guid} +1 and p_{init} in AL_{cu} are moved right before q in the same order.

The resulting AL_{cu} is added to *PR_set*. We then set $AL_{init} = AL_{guid}$ and $AL_{guid} = AL_{cu}$ and repeat the process until $AL_{cu} = AL_{guid}$, thus connecting AL_1 and AL_2 with a single path composed of all the activity lists stored in *PR_set*. Only a subset of *PR_set* will be selected for evaluation with the serial SGS. Let n_{pr} be the number of activity lists to be selected for evaluation. The activity lists in *PR_set* are numbered from 1 to |*PR_set*| and placed in n_{pr} subsets of size |*PR_set*| / n_{pr}. One AL from each subset is randomly selected and evaluated to find its schedule and its makespan.

3.8. Improvement strategy - FBI

Search intensification is typically achieved in SS with the execution of an improvement method applied on the new solutions found during the combination process. As pointed by Marti et al. (2006), an important issue in SS design is how to allocate the computational effort between improving current solutions and generating new ones. In the existing literature on SS applied to RCPSP, authors have given various levels of priority to improvement. At the one extreme, Ranjbar et al. (2009) did not apply any improvement method. At the other extreme, Paraskevopoulos et al. (2012) introduced a local search and a perturbation strategy based on a long-term memory, while Valls et al. (2004) and Chen et al. (2010) applied a procedure that improves the local resource utilization until no further improvement can be produced. In between, Debels et al. (2006) and Mobini et al. (2009) applied the well-known FBI procedure. Introduced by Li and Willis (1992) and popularized by Tormos and Lova (2001) and Valls et al. (2005), FBI iteratively applies forward and backward passes until no further improvement can be produced. In the forward (backward) pass, the activities are scheduled as early as possible (as late as

possible). As this simple technique produces notable improvements in schedules quality with a small computation effort (Valls et al. 2005), it is introduced in the proposed SS.

In the first phase of the FBI, the backward is done by applying the transformation from a AL to a TOAL to obtain a reverse (direct) TOAL from a direct (reverse) AL, reversing the project network from direct (reverse) to a reverse (direct) network, and then scheduling the TOAL with the reverse (direct) network. In the second phase, the forward pass is applied in the same manner to obtain a direct (reverse) schedule in the initial direct (reverse) network direction. As proved in Valls et al. (2005), the makespan of the schedule after an iteration of FBI is inferior or equal to the initial makespan. This process is repeated if the makespan has been improved. In the proposed SS, FBI is first applied on the best solutions of the initial population, and then only on high-quality solutions that match or improve the best makespan found so far during the process.

3.9. New Population generation

The new population is first initialized with the best solution found so far. In case of tie, the rule proposed in section 3.5 is considered. For each subset, n_{pr} activity lists are generated with PR, scheduled with the serial SGS and possibly improved by FBI. This constitutes the new population for the next SS iteration, thus composed of $\left(\frac{b_1 \cdot (b_1 - 1)}{2} + b_1 \cdot b_2\right) \cdot n_{pr} + 1$ solutions.

4. Computational analysis

This section presents an evaluation of the proposed SS on the PSPLIB data sets proposed in Kolisch and Sprecher (1996), and a comparison with the state-of-the-art heuristics developed for the RCPSP. The input parameters of the SS algorithm are first determined empirically by parameter tuning. The experiments were conducted on a personal computer with an Intel Core I5 2.53 GHz processor and 4 GB RAM under Windows 7 Professional. The algorithm was implemented in Matlab R2011b.

4.1. PSPLIB data sets and test design

As test instances, we use the standard J30, J60 and J120 sets of the PSPLIB library, which have been generated using PROGEN (Kolisch et al. 1992, 1995). The J30 and J60 sets consist of 480 projects of 30 and 60 activities, respectively. The J120 set is composed of 600 projects of 120 activities. For more details on how the projects data are generated, the reader is referred to Kolisch et al. (1992, 1995). The complete data are available from the project scheduling library PSPLIB on the internet. Even though other benchmark instances exist in the literature, such as the Patterson instances (Patterson, 1984), the PSPLIB library is the most widely used for experimental comparison of heuristics for the RCPSP.

An experimental protocol for testing the heuristics with the PSPLIB instances has been proposed by Hartmann and Kolisch (2000) and Kolisch and Hartmann (2006). Three stopping criteria are considered: 1000, 5000 and 50000 generated schedules. Since the speed of computers has increased and since some researchers have observed significant improvement in the results with larger schedule limits, our method is also tested with 500000 schedules for J60 and J120. A schedule-based stopping criterion is used because the corresponding computational effort is independent from the evolution of the speed of computers, the operating systems, the compilers and the implementational skills and is then quite similar for all the tested heuristics (Kolisch and Hartmann, 2006). The heuristics are then compared according to the average deviation from the optimal solutions (for J30) or from the well-known critical path-based lower bound (for J60 and J120).

4.2. Impact of randomness on the performance

As for most metaheuristic approaches used to solve the RCPSP, many parts of the proposed SS are based on random devices. In our algorithm, randomness is involved in the initial population generation, the selection of the n_{pr} activity lists from *PR_set*, and the diversification to complete $RefSet_1$ and $RefSet_2$ when required. Each execution for the same problem instance would provide different results when random devices are used. However, most papers on metaheuristic approaches for the RCPSP do not indicate if multiple executions are conducted. In Hartmann and Kolisch (2000), Kolisch and Hartmann (2006) and in the papers published afterwards, the performance measure used for sorting the metaheuristics is the average deviation measured with a single figure in percentage with a two-digit precision. However, the value of the schedule limits were arbitrarily set in order to provide a unified base of comparison of the performances of heuristics, but they do not guarantee a convergence of the results and a margin of errors due to randomness in accordance with a two-digit precision. In order to take into account the effect of randomness in our experiments, each instance is solved ten times and the average results of the ten replications, as well as the 95% confidence interval, are provided.

4.3. Parameter tuning

The proposed SS is based on six input parameters: *InitPop*, b_1, b_2, t_1, t_2 and n_{pr}. Preliminary tests showed that the performances of the SS are highly sensitive to the values of these parameters. Thus, their values have to be carefully set with parameter tuning. Among the method used for varying the parameters in the literature, most authors used full factorial or Taguchi designs of experiment, or elementary trial and error strategies. Tuning is usually performed on the whole standard sets, or a subset of instances or a new sample of instances. Finally, fine tuning can be conducted for each instance size, for each schedule limit or each combination of schedule limit and project size. Based on preliminary tests, we noted that the promising combinations had

different parameters values depending on the schedule limit and the project size. Therefore, fine tuning was independently performed on each combination of schedule limit and project size. We also observed that a full factorial design with six parameters would not be efficient to explore the space of combinations and would be time consuming, unless the subsets used for tuning are very small or unless the number of repetitions is reduced. However, a weak correlation was observed between the performances for very small subsets of instances or for few repetitions and the performances on the whole standard sets with ten repetitions.

Consequently, we propose a new approach for parameter tuning that consists in a basic local search process which moves from a combination of parameters to another. Starting from an initial combination of parameters, each parameter is changed to a higher or a lower value with a predefined step, one at a time. The combination with the lowest average deviation in this neighborhood is used as starting combination for the next iteration. To avoid being trapped in local optima, we authorize non-improving iterations. Visited combinations are memorized to avoid cycling and to restart the process from the best not selected combinations after a number of non-improving iterations. The local search is stopped after a number of non-improving iterations, sometimes after several hundreds of tested combinations.

This method is applied to find the best combinations of parameters for each standard set (J30, J60, J120) and each schedule limit (1000, 5000, 50000, 500000 schedules). The local search is applied on the whole standard sets and ten independent runs are conducted, except for the J60 set with 500000 schedules and for the J120 set with 50000 and 500000 schedules. For these combinations, the number of instances and the number of independent runs are reduced. For instance, only five independent runs are conducted on a subset of the hardest instances for J120 with 50000 schedules (i.e., the instances with a deviation higher than 4% between the makespan obtained with 5000 schedules and the best known makespan in literature). Table 1 presents the best combinations of parameters for each instance set and each schedule limit.

Instance set	J30			J60				J120			
schedule limit	1000	5000	50000	1000	5000	50000	500000	1000	5000	50000	500000
initPop	100	100	1000	100	200	1500	7500	100	100	1100	3000
b_1	8	15	35	7	10	27	75	5	9	17	40
b_2	4	16	29	4	9	21	65	4	7	17	40
t_1	0.8	0.6	0.5	0.8	0.9	1.3	1.2	0.8	1.7	2.2	2.3
t_2	1.5	1.5	1.4	1.6	1.8	2.1	2.3	1.9	2.7	3.8	3.7
n_{pr}	1	1	1	1	1	1	1	1	1	1	1

Table 1 Parameter tuning

4.4. Detailed results of the scatter search algorithm

The detailed experimental results are presented in Table 2. The row labeled “Sum” gives the average sum of the makespans of all the instances of the set with ten independent runs. The row labeled “Avg. dev. CPM”, “Avg. dev. LB” and “Avg. dev. best” indicate the average deviation from

Instance set	J30			J60				J120			
schedule limit	1000	5000	50000	1000	5000	50000	500000	1000	5000	50000	500000
Sum	28354	28324	28316	38647	38495	38374	38333	76125	75205	74437	74001
Avg. dev. CPM	13.53%	13.40%	13.37%	11.38%	10.93%	10.58%	10.45%	34.13%	32.52%	31.16%	30.39%
Avg. dev. LB	0.10%	0.02%	0.00%	1.33%	1.02%	0.77%	0.68%	6.27%	5.18%	4.29%	3.78%
Avg. dev. best	0.10%	0.02%	0.00%	0.68%	0.38%	0.13%	0.05%	3.09%	2.06%	1.22%	0.74%
No. of instances	480	480	480	480	480	480	480	600	600	600	600
Avg. No. of best	452	474	480	371	392	427	454	206	230	260	301
Avg. CPU (s)	0.27	0.80	3.50	0.90	3.24	34.66	735.60	4.62	21.04	198.14	2976.53
Max. CPU (s)	2.80	15.19	191.02	5.32	26.42	369.89	12800.24	10.69	83.93	647.67	14179.8
Avg. No. sched.	100	248	760	176	610	4578	26979	457	1917	15699	117526

Table 2 Detailed experimental results for ten repetitions

the critical path lower bound, the best known lower bound and the current best known makespan, respectively. For the latter value, the reader is referred to the PSPLIB website which indicates for each instance the best known makespan and the author who found the solution. The results available on January 1, 2014 were used. “No. of instances” represents the number of instances in the set, while “Av. No. of best” is the number of instances for which the makespan obtained with the proposed method is as good as the current best known makespan. “Avg. CPU” and “Max. CPU” indicate the average and maximum computation times to reach the best solution, respectively. “Avg. No. sched.” represents the average number of generated schedules to reach the best solution.

Table 2 highlights that the algorithm is able to find all the optimal solutions of J30 with 50000 schedules in reasonable computation times. Near-optimal solutions can even be found with 1000 and 5000 schedules for J30 in very small computation times. For J60, our method provides an average deviation of 0.05 % with the best known makespans for 500000 schedules, and reaches the best solutions in 454 instances out of 480. Similarly, we obtain for J120 and 500000 schedules an average deviation of 0.74% with the best known makespans, reaching the best solutions in 301 instances out of 600. This is remarkable as the best known makespans have been obtained with any stopping criterion and any existing heuristic in the literature, and as there is currently no unique method that can reach all the best known solutions for J60 and J120. In addition, the proposed SS is able to find attractive solutions with lower schedule limits. For instance, we obtain an average deviation of only 0.68% with the best known makespans for J60 with 1000 schedules in less than 1 second.

Table 3 summarizes the detailed results for each repetition ("Dev. (%) rep."), as well as the average deviation for ten repetitions ("Avg. Dev. (%)"), the minimum and maximum deviation observed ("Min. Dev. (%)" and ("Max. Dev. (%)", resp.) and the 95% confidence interval ("95% Conf. int. (%)"). It reveals that the larger the size of the project or the lower the schedule limit, the wider the 95 % confidence interval.

Instance set	J30			J60				J120			
schedule limit	1000	5000	50000	1000	5000	50000	500000	1000	5000	50000	500000
Dev. (%) rep. 1	0.11	0.02	0.00	11.39	10.94	10.57	10.46	34.16	32.50	31.15	30.37
Dev. (%) rep. 2	0.11	0.02	0.00	11.39	10.93	10.58	10.45	34.25	32.51	31.12	30.37
Dev. (%) rep. 3	0.09	0.03	0.00	11.43	10.95	10.60	10.46	34.15	32.54	31.15	30.36
Dev. (%) rep. 4	0.09	0.02	0.00	11.43	10.95	10.59	10.46	34.14	32.54	31.16	30.40
Dev. (%) rep. 5	0.11	0.01	0.00	11.34	10.89	10.58	10.45	34.07	32.46	31.19	30.39
Dev. (%) rep. 6	0.10	0.02	0.00	11.34	10.89	10.57	10.45	34.13	32.53	31.16	30.44
Dev. (%) rep. 7	0.10	0.01	0.00	11.36	10.96	10.57	10.45	34.14	32.51	31.20	30.39
Dev. (%) rep. 8	0.10	0.02	0.00	11.44	10.96	10.56	10.46	34.09	32.48	31.14	30.44
Dev. (%) rep. 9	0.11	0.03	0.00	11.36	10.92	10.56	10.45	34.03	32.56	31.17	30.39
Dev. (%) rep. 10	0.12	0.02	0.00	11.38	10.95	10.58	10.44	34.19	32.54	31.19	30.40
Avg. Dev. (%)	0.10	0.02	0.00	11.38	10.93	10.58	10.45	34.13	32.52	31.16	30.39
Min. Dev. (%)	0.09	0.01	0.00	11.34	10.89	10.56	10.44	34.03	32.46	31.12	30.36
Max. Dev. (%)	0.12	0.03	0.00	11.44	10.96	10.60	10.46	34.25	32.56	31.20	30.44
95% Conf. int. (%)	[0.10; 0.11]	[0.02; 0.02]	[0.00; 0.00]	[11.36; 11.41]	[10.92; 10.95]	[10.57; 10.58]	[10.45; 10.46]	[34.10; 34;17]	[32.50; 32.54]	[31.15; 31.18]	[30.38; 30.41]
Width of the 95% Conf. int. (%)	0.01	0.01	0.00	0.05	0.03	0.02	0.01	0.08	0.04	0.03	0.04

Table 3 Overview of the average deviation for ten repetitions

4.5. Comparative analysis with the best metaheuristic approaches

The comparative analysis is divided into two sections. In the first section, the results of the proposed approach are compared with the best heuristics that provided results following the experimental protocol summarized in section 4.1. In the second section, we compare the results of the heuristics that do not use any schedule limit.

4.5.1. Comparative analysis with schedule limits

For reasons of space, the comparison is limited to the twenty best heuristics with 50000 schedules for either J30, J60 or J120. From the state-of-the-art heuristics surveyed in Kolisch and Hartmann (2006), only the heuristics proposed by Kochetov and Stolyar (2003), Alcaraz et al. (2004), Valls et al. (2005) and Debels et al. (2006), which were all identified as the best heuristics at this time, are still currently in this list. This highlights how fruitful the recent developments in heuristics for the RCPSP are. The computational results of the selected heuristics can be found in Tables 4, 5 and 6 for the instance sets J30, J60, and J120, respectively. All these methods are metaheuristics. As explained before, most of these metaheuristics are stochastic, leading to different results at each execution. However, most authors did not indicate if their results were obtained with a single run, or with several runs, and, in the latter case, if their results were the

best or the average among several runs. In order to compare our results, the average deviation observed with ten independent runs of the proposed SS algorithm is considered.

The first column of Tables 4‑6 briefly describes each heuristic with the abbreviations used by their authors or commonly used in literature: GA for Genetic Algorithm, SAILS for Scatter Search

Algorithm	SGS	Authors	Average Deviation (%)			>50000	
			1000	5000	50000	Avg. dev. (%)	No. of sched.
Best known makespans	—	—		**0**			
SAILS	Serial	Paraskevopoulos et al. (2012)	0.03	0.01	0		
SS—FBI	**Serial**	**This study**	**0.10**	**0.02**	**0**		
Artificial Immune Algo.—FBI	Serial	Mobini et al. (2011)	0.05	0.03	0		
SS—FBI	Serial	Ranjbar et al. (2009)	0.10	0.03	0		
GA, TS—PR	Both	Kochetov and Stolyar (2003)	0.10	0.04	0		
GA—FBI	Both	Wang et al. (2010)	0.14	0.04	0		
GA	Both	Zamani (2013)	0.14	0.04	0		
Bees Algo.	—	Sadeghi et al. (2011)	0.15	0.09	0		
ESS	Both	Mobini et al. (2009)	0.05	0.02	0.01		
GA	Both	Mendes et al. (2009)	0.06	0.02	0.01		
GA—FBI	Serial	Gonçalves et al. (2011)	0.32	0.02	0.01	0.01	499860
GA—DP	Both	Cervantes et al. (2008)	0.16	0.04	0.01		
PSO—HH	Serial	Koulinas et al. (2014)	0.26	0.04	0.01		
DABC	Serial	Nouri et al. (2013)	0.21	0.05	—		
ACOSS	Both	Chen et al. (2010)	0.14	0.06	0.01		
Cooling Process—GA—FBI	Serial	Lim et al. (2013)	0.21	0.07	0.01	0.00	500000
SS—FBI	Serial	Debels et al. (2006)	0.27	0.11	0.01	0.01	500000
GA—FBI	Serial	Debels and Vanhoucke (2007)	0.15	0.04	0.02		
GA—hybrid, FBI	Serial	Valls et al. (2008)	0.27	0.06	0.02		
Memetic Algo. —FBI	Serial	Carlier et al. (2009)	0.32	0.11	0.02		
GA—FBI	Serial	Valls et al. (2005)	0.34	0.20	0.02		
GA—FBI	Both	Alcaraz et al. (2004)	0.25	0.06	0.03		
SFLA—FBI	Serial	Fang and Wang (2012)	0.36	0.21	0.18		
GANS	Serial	Proon and Jin (2011)	1.83	1.27	0.71		

Table 4 Computational comparison with schedule limits - J30

with Adaptive Iterated Local Search, ESS for enhanced Scatter Search, TS for Tabu Search, PSO-HH for Particle Swarm Optimization based Hyper-Heuristic algorithm, DABC for Discrete Artificial Bee Colony algorithm, ACOSS for Ant Colony Optimization and SS, SFLA for Shuffled Frog-leaping Algorithm, GANS for GA with Neighborhood Search. The first line “Best known makespans” provides the average deviation based on the current best known makespans obtained with any heuristic and any stopping criterion (see section 4.4). The next column indicates the type of SGS employed. The average deviations for 1000, 5000 and 50000 schedules appears in the fourth, fifth and sixth columns, respectively. For each instance set, the heuristics are sorted in ascending order according to the average deviation with 50000 schedules. In case of tie, the results for 5000 and then 1000 schedules are considered. The

seventh and eighth columns indicate the average deviation and the schedule limit for the heuristics tested with more than 50000 schedules.

As highlighted in Table 4, the proposed SS is only slightly outperformed for J30 by the method of Paraskevopoulos et al. (2012). Our approach belongs to the metaheuristics able to reach all the optimal solutions with 50000 schedules. If we consider the best algorithms with 5000 or 50000

Algorithm	SGS	Authors	Average Deviation (%)			>50000	
			1000	5000	50000	Avg. dev. (%)	No. of sched.
Best known makespans	—	—		**10.37**			
GANS	Serial	Proon and Jin (2011)	11.35	10.53	10.52		
SAILS	Serial	Paraskevopoulos et al. (2012)	11.05	10.72	10.54	10.46	70000
Artificial Immune Algo.—FBI	Serial	Mobini et al. (2011)	11.17	10.80	10.55		
ESS	Both	Mobini et al. (2009)	11.12	10.74	10.57		
GA—FBI	Both	Wang et al. (2010)	11.55	10.96	10.57		
GA—FBI	Serial	Gonçalves et al. (2011)	—	11.56	10.57	10.49	499140
SS—FBI	**Serial**	**This study**	**11.38**	**10.93**	**10.58**	**10.45**	**500000**
Cooling Process—GA—FBI	Serial	Lim et al. (2013)	11.73	11.14	10.63	10.51	500000
SS—FBI	Serial	Ranjbar et al. (2009)	11.59	11.07	10.64		
GA	Both	Zamani (2013)	11.33	10.94	10.65		
SFLA—FBI	Serial	Fang and Wang (2012)	11.44	10.87	10.66		
ACOSS	Both	Chen et al. (2010)	11.75	10.98	10.67		
GA	Both	Mendes et al. (2009)	11.72	11.04	10.67	10.67	63546
GA	Serial	Debels and Vanhoucke (2007)	11.45	10.95	10.68		
PSO—HH	Serial	Koulinas et al. (2014)	11.74	11.13	10.68		
Memetic Algo. —FBI	Serial	Carlier et al. (2009)	11.62	11.09	10.70		
SS—FBI	Serial	Debels et al. (2006)	11.73	11.10	10.71	10.53	500000
GA—hybrid, FBI	Serial	Valls et al. (2008)	11.56	11.10	10.73		
GA, TS—PR	Both	Kochetov and Stolyar (2003)	11.71	11.17	10.74		
GA—FBI	Serial	Valls et al. (2005)	12.21	11.27	10.74		
Bees Algo.	—	Sadeghi et al. (2011)	11.93	11.48	10.74		
GA—DP	Both	Cervantes et al. (2008)	11.43	10.96	10.81		
DABC	Serial	Nouri et al. (2013)	11.74	11.16	—		
GA—FBI	Both	Alcaraz et al. (2004)	11.89	11.19	10.84		

Table 5 Computational comparison with schedule limits - J60

schedules, most of them have a difference of less than 0.01 point from each other. Assuming that the width of the 95% confidence interval for any metaheuristic has the same order of magnitude as the one observed in Table 3 for our SS (0.01 point for 1000 and 5000 schedules, 0.00 point for 50000 schedules), we can conclude that the precision used by researchers is appropriate and that randomness does not significantly change the current ranking.

Table 5 shows that none of the heuristics is able to find all the best known solutions for J60. The proposed algorithm is ranked 7th for 50000 schedules based on the average deviation. However, only three algorithms in the literature are better than the lower bound of the 95% confidence interval presented in Table 3 (10.57%). The proposed SS is ranked 6th for 1000 and 5000 schedules based on the average deviation, indicating that it is also competitive with lower schedule limits. As for J30, the average deviations obtained by the best metaheuristics for 50000

schedules are also close to each other. However, assuming that the width of 95% confidence interval for any metaheuristic has the same order of magnitude as the one observed in Table 3 (0.05, 0.03 and 0.02 point for 1000, 5000 and 50000, resp.), it may be inappropriate to rank the methods with the precision commonly used by researchers. It would be more appropriate to gather them in groups where the difference in performances could be explained by randomness. With this framework, Proon and Jin (2011) developed the best metaheuristic, followed by a

Algorithm	SGS	Authors	Average Deviation (%)			>50000	
			1000	5000	50000	Avg. dev. (%)	No. of sched.
Best known makespans	—	—		**29.18**			
GANS	Serial	Proon and Jin (2011)	33.45	31.51	30.45		
ACOSS	Both	Chen et al. (2010)	35.19	32.48	30.56		
Cooling Process—GA—FBI	Serial	Lim et al. (2013)	34.95	32.75	30.66	29.91	500000
SAILS	Serial	Paraskevopoulos et al. (2012)	33.32	32.12	30.78	30.39	200000
GA	Serial	Debels and Vanhoucke (2007)	34.19	32.34	30.82		
SFLA—FBI	Serial	Fang and Wang (2012)	34.83	33.20	31.11		
SS—FBI	**Serial**	**This study**	**34.13**	**32.52**	**31.16**	**30.39**	**500000**
PSO—HH	Serial	Koulinas et al. (2014)	35.20	32.59	31.23		
GA—hybrid, FBI	Serial	Valls et al. (2008)	34.07	32.54	31.24	30.95	100000
GA—FBI	Both	Wang et al. (2010)	35.18	33.11	31.28		
GA	Both	Zamani (2013)	34.02	32.89	31.30		
ESS	Both	Mobini et al. (2009)	34.51	32.61	31.37	31.20	127341
GA	Both	Mendes et al. (2009)	35.87	33.03	31.44		
Artificial Immune Algo.—FBI	Serial	Mobini et al. (2011)	34.01	32.57	31.48		
Memetic Algo. —FBI	Serial	Carlier et al. (2009)	34.89	33.18	—		
SS-FBI	Serial	Ranjbar et al. (2009)	35.08	33.24	31.49		
DABC	Serial	Nouri et al. (2013)	36.40	33.72	31.49		
GA—FBI	Both	Alcaraz et al. (2004)	36.53	33.91	31.49		
Bees Algo.	—	Sadeghi et al. (2011)	35.80	33.33	31.55		
SS—FBI	Serial	Debels et al. (2006)	35.22	33.10	31.57	30.48	500000
GA—FBI	Serial	Valls et al. (2005)	35.39	33.24	31.58		
GA—DP	Both	Cervantes et al. (2008)	33.71	32.57	31.65		
GA, TS—PR	Both	Kochetov and Stolyar (2003)	34.74	33.36	32.06		
GA—FBI	Serial	Gonçalves et al. (2011)	—	35.94	32.76	30.08	490320

Table 6 Computational comparison with schedule limits - J120

group composed of Paraskevopoulos et al. (2012) and Mobini et al. (2011), and another group composed of this paper, Mobini et al. (2009), Wang et al. (2010) and Gonçalves et al. (2011). To further illustrate the impact of randomness on the performances of the proposed SS, note that with *initPop* = 1500 b_1 = 27, b_2 = 21, t_1 =1.1, t_2 = 2.1, n_{pr} = 1, the minimal average deviation among ten runs is 10.52%, which would match the results obtained by Proon and Jin (2011), but the maximal measure is 10.62%, leading to an average value of 10.59%.

For J120, the results in Table 6 are more spread than for J30 and J60. As for J60, none of the existing heuristics is able to find all the best known solutions. The proposed SS is ranked 7th, 5th and 6th for 50000, 5000 and 1000 schedules, respectively. For this instance set, the width of the

confidence interval that we observed in Table 3 for the proposed SS (0.08, 0.04 and 0.03 point for 1000, 5000 and 50000 schedules, respectively) does not change the ranking.

Some papers conducted experiments for more than 50000 schedules. As shown in Tables 5-6, the results are significantly improved for J60 and J120, which should incite researchers to test metaheuristics for larger schedule limits. As these algorithms do not use the same schedule limit, the comparison of the results should be taken with caution. The proposed SS performs better than any other metaheuristics for J60, while the obtained results are close to those obtained in the literature for J120.

As presented in Tables 4-6, no heuristic provides the best results for all the schedule limits and all the project sizes. For that reason, Kolisch and Hartmann (2006) introduced the concept of dominance in order to determine the best global heuristics. A heuristic X is dominated by a heuristic Y if X has for at least one combination of instance set and schedule limit a higher average deviation than Y without having for any of the other combinations a lower average deviation. By applying this rule, the proposed SS would only be dominated by Paraskevopoulos et al. (2012) for 1000, 5000 and 50000 schedules, which makes it one of the best heuristics and confirms that hybrid metaheuristics, especially those based on scatter search, are among the best current heuristics. However, Paraskevopoulos et al. (2012) do not indicate if their results were obtained with a single run or with several runs, and, in the latter case, if their results were the best or the average among several runs. As previously explained for J60, our method is hence able to provide better results than Paraskevopoulos et al. (2012) for a single run with a schedule limit of 50000, but worst results for ten replications.

4.5.2. Comparative analysis with no schedule limits

Some researchers have not provided results according to schedule limits, as it is sometimes impossible to count the number of generated schedules. These methods are presented in Tables 7-9 for J30, J60 and J120, respectively. The first column describes each heuristic: PCAP for Parallel Complete Anytime Procedure, FF for Filter-and-Fan approach, LSSPER for Local Search with Subproblem Exact Resolution, VNS for Variable Neighborhood Search, PBP for Population Based Procedures, DBGA for Decomposition Based GA, MP for Multi-Pass approach, SA for Simulated Annealing, ATLAS for Accelerating Two-Layer Anchor Search, PASS for Polarized Adaptive Scheduling Scheme, LR for Lagrange Relaxation. Since the reported results differ in terms of stopping criterion and computational effort, it is hard to carry out any comparison with the proposed SS. The analysis presented hereinafter should then be taken with caution.

For J30, Table 7 shows that the proposed SS produces better results in less time than these algorithms, except the GA of Hindi et al. (2002), the VNS of Fleszar and Hindi (2004) and the DBGA of Debels and Vanhoucke (2007) and. The latter achieved a worse average deviation than

the one obtained with the proposed SS for 1000 schedules, but with a significantly faster computation time. For 5000 schedules, Fleszar and Hindi (2004) obtained a better average deviation with a slightly lower computation time. As presented in Table 8 and Table 9, Thammano and Phu-ang (2012) obtained competitive results but they did not provide the computational times and the numbers of generated schedules. For J60, Ranjbar (2008) produced a better average deviation than the proposed SS for 50000 schedules, with a better computational time. Debels and Vanhoucke (2007) and Zamani (2012) obtained a worse average deviation compared to the proposed SS for 50000 schedules, but with better computational times. Valls et al. (2004) obtained a slightly better average deviation with a slightly faster computation time. Table 9 highlights that Debels and Vanhoucke (2007) produced better results than the proposed SS for J120 and 50000 schedules in terms of average deviation and computation time, while Valls (2004), Ranjbar (2008) and Zamani (2012) produced competitive results very fast. From this analysis, it can be concluded that none of these heuristics globally perform better than the proposed SS in terms of both computation time and solution quality for all the instance sets.

Algorithm	SGS	Authors	Av. dev. (%)	No. of schedules		CPU(s)		CPU freq.
				Av.	Max.	Av.	Max.	
SS—FBI	**Serial**	**This study (50000 sched.)**	**0.00**	**—**	**—**	**3.50**	**191.02**	**2.53 GHz**
PCAP	—	Zamani (2010b)	0.00	—	—	4.09	—	1.86 GHz
FF	Serial	Ranjbar (2008)	0.00	—	—	5.00	—	3 GHz
LSSPER	Serial	Palpant (2004)	0.00	830	1120	10.26	123.00	2.3 GHz
VNS	Serial	Fleszar and Hindi (2004)	0.01	—	—	0.64	5.86	1 GHz
SS—FBI	**Serial**	**This study (5000 sched.)**	**0.02**	**—**	**—**	**0.80**	**15.19**	**2.53 GHz**
TS—FBI	Serial	Valls et al. (2003)	0.06	—	—	1.61	6.15	400 MHz
SS—FBI	**Serial**	**This study (1000 sched.)**	**0.10**	**—**	**—**	**0.27**	**2.80**	**2.53 GHz**
PBP	Serial	Valls et al. (2004)	0.10	—	—	1.16	5.49	400 MHz
DBGA	Serial	Debels and Vanhoucke (2007)	0.12	—	—	0.01	—	1.8 GHz
Netw. Decomp.	—	Sprecher (2002)	0.12	—	—	2.75	39.7	166 MHz
GA	Serial	Hindi et al. (2002)	0.37	1683	3068	0.17	0.49	400 MHz
MP—netw. flow	Paral.	Artigues et al. (2003)	1.74	30	30	—	—	—

Table 7 Computational comparison with an unlimited number of schedules - J30

Algorithm	SGS	Authors	Av. dev. (%)	No. of schedules		CPU(s)		CPU freq.
				Av.	Max.	Av.	Max.	
SS—FBI	**Serial**	**This study (500000 sched.)**	**10.45**	**—**	**—**	**735.60**	**12800.24**	**2.53 GHz**
GA—TS—SA	Serial	Thammano and Phu-ang (2012)	10.52	—	—	—	—	—
FF	Serial	Ranjbar (2008)	10.56	—	—	5.00	—	3 GHz
SS—FBI	**Serial**	**This study (50000 sched.)**	**10.58**	**—**	**—**	**34.66**	**369.89**	**2.53 GHz**
PASS	Both	Zamani (2012)	10.64	—	—	3.00	—	1.86 GHz
DBGA	Serial	Debels and Vanhoucke (2007)	10.68	—	—	1.11	—	1.8 GHz
ATLAS	Both	Zamani (2010a)	10.81	—	—	36.20	—	1.86 GHz
LSSPER	Serial	Palpant (2004)	10.81	1622	2262	38.78	223	2.3 GHz
PBP	Serial	Valls et al. (2004)	10.89	—	—	3.65	22.60	400 MHz
SS—FBI	**Serial**	**This study (5000 sched.)**	**10.93**	**—**	**—**	**3.24**	**26.42**	**2.53 GHz**

VNS	Serial	Fleszar and Hindi (2004)	10.94	—	—	8.89	80.70	1 GHz
PCAP	—	Zamani (2010b)	11.07	—	—	28.45	—	1.86 GHz
SS—FBI	**Serial**	**This study (1000 sched.)**	**11.38**	**—**	**—**	**0.90**	**5.32**	**2.53 GHz**
TS—FBI	Serial	Valls et al. (2003)	11.45	—	—	2.76	14.61	400 MHz
Netw. Decomp.	—	Sprecher (2002)	11.61	—	—	460.16	4311.46	166 MHz
TS—netw. flow	Paral.	Artigues et al. (2003)	12.05	—	—	3.20	—	—
MP—netw. flow	Paral.	Artigues et al. (2003)	14.20	60	60	—	—	—
LR—activity list	Both	Möhring et al. (2003)	15.60	—	—	6.9	57	200 MHz

Table 8 Computational comparison with an unlimited number of schedules - J60

Algorithm	SGS	Authors	Av. Dev. (%)	No. of schedules		CPU(s)		CPU freq.
				Av.	Max.	Av.	Max.	
SS—FBI	**Serial**	**This study (500000 sched.)**	**30.39**	**—**	**—**	**2976.53**	**14179.87**	**2.53 GHz**
GA—TS—SA	Serial	Thammano and Phu-ang (2012)	30.51	—	—	—	—	—
DBGA	Serial	Debels and Vanhoucke (2007)	30.69	—	—	2.99	—	1.8 GHz
SS—FBI	**Serial**	**This study (50000 sched.)**	**31.16**	**—**	**—**	**198.14**	**647.67**	**2.53 GHz**
PASS	Both	Zamani (2012)	31.22	—	—	8.00	—	1.86 GHz
FF	Serial	Ranjbar (2008)	31.42	—	—	5.00	—	3 GHz
PBP	Serial	Valls et al. (2004)	31.58	—	—	59.43	263.97	400 MHz
LSSPER	Serial	Palpant (2004)	32.41	3396	5000	207.93	501.00	2.3 GHz
ATLAS	Both	Zamani (2010a)	32.45	—	—	110.50	—	1.86 GHz
SS—FBI	**Serial**	**This study (5000 sched.)**	**32.52**	**—**	**—**	**21.04**	**83.93**	**2.53 GHz**
VNS	Serial	Fleszar and Hindi (2004)	33.10	—	—	219.86	1126.97	1 GHz
PCAP	—	Zamani (2010b)	33.78	—	—	34.23	—	1.86 GHz
SS—FBI	**Serial**	**This study (1000 sched.)**	**34.13**	**—**	**—**	**4.62**	**10.69**	**2.53 GHz**
TS—FBI	Serial	Valls et al. (2003)	34.53	—	—	17.00	43.94	400 MHz
LR—activity list	Both	Möhring et al. (2003)	36.00	—	—	72.90	654.00	200 MHz
TS—netw. flow	Paral.	Artigues et al. (2003)	36.16	—	—	67.00	—	—
Netw. Decomp.	—	Sprecher (2002)	39.29	—	—	458.53	1511.33	166 MHz
MP—netw. flow	Paral.	Artigues et al. (2003)	39.34	120	120	—	—	—

Table 9 Computational comparison with an unlimited number of schedules - J120

5. Concluding remarks

This paper presents a new hybrid metaheuristic for solving the resource-constrained scheduling project problem (RCPSP). The method is based on a scatter search framework that involves forward-backward improvement, reversing of population at each iteration of the scatter search and an advanced path relinking strategy for combining pairs of solutions. This path relinking has two distinctive features. First, it is a bidirectional path relinking, which has been identified as one of the most promising approach for combination in scatter search. It consists in swaping the initiating and the guiding solutions at each step of the path relinking process. Second, this path relinking is based on a new move that consists in moving the most distant activity from the guiding solution in the current solution of the path to its place in the guiding solution.

Since the performances are highly sensitive to the values of the scatter search parameters, an advanced parameter tuning method based on local search is proposed. Computational experiments on the PSPLIB benchmark (Kolisch and Sprecher 1996) show that the proposed scatter search algorithm provides high-quality solutions in reasonable computation times. An

extensive comparison with the best heuristics in the literature demonstrates that the proposed approach is one of the most advanced heuristics, especially for the J30 and J60 instance sets.

We suggest three directions for further research. First, based on the obtained results, the experimental protocol commonly used in the literature should be updated by conducting several independent runs to include the effect of random devices involved in most metaheuristics, and by testing the heuristics on larger problem instances and for larger schedule limits. Second, since the hybrid metaheuristics based on scatter search are among the best heuristics, further research should be oriented to develop new mechanisms for scatter search and to include existing advanced ones for the schedule representation, the decoding procedure, the generation of the initial population, the subset generation method, the combination method, the improvement method and the use of long-term memory. Finally, the proposed hybrid scatter search could be used to tackle other combinatorial optimization problems including variants of the standard RCPSP.

Acknowledgements

This work has been supported by the Natural Sciences and Engineering Research Council of Canada, the Jarislowsky/SNC-Lavalin Research Chair in the Management of International Projects, and the CIRRELT. This support is gratefully acknowledged.

References

Alcaraz, J., Maroto, C., & Ruiz, R. (2004). *Improving the performance of genetic algorithms for the RCPS problem*. Proceedings of the Ninth International Workshop on Project Management and Scheduling PMS 2004, Nancy, France (p. 40-43).

Artigues, C., Michelon, P., & Reusser, S. (2003). Insertion techniques for static and dynamic resource-constrained project scheduling. *European Journal of Operational Research*, *149*(2), 249-267.

Blazewicz, J., Lenstra, J.K., & Kan, A.H.G. (1983). Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*, *5*(1), 11-24.

Brucker, P., Drexl, A., Möhring, R., Neumann, K., & Pesch, E. (1999). Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, *112*(1), 3-41.

Carlier, J., Moukrim, A., & Xu, H. (2009). *A Memetic Algorithm for the Resource Constrained Project Scheduling Problem*. In Proceedings of the International Conference on Industrial Engineering and Systems Management IESM 2009, Montreal, Canada.

Cervantes, M., Lova, A., Tormos, P., & Barber, F. (2008). A dynamic population steady-state genetic algorithm for the resource-constrained project scheduling problem. In *New Frontiers in Applied Artificial Intelligence* (p. 611-620). Berlin Heidelberg, Germany: Springer.

Chen, W., Shi, Y.J., Teng, H.F., Lan, X.P., & Hu, L.C. (2010). An efficient hybrid algorithm for resource-constrained project scheduling. *Information Sciences*, *180*(6), 1031-1039.

Debels, D., De Reyck, B., Leus, R., & Vanhoucke, M. (2006). A hybrid scatter search/electromagnetism metaheuristic for project scheduling. *European Journal of Operational Research*, 169(2), 638-653.

Debels, D., & Vanhoucke, M. (2007). A decomposition-based genetic algorithm for the resource-constrained project-scheduling problem. *Operations Research*, *55*(3), 457-469.

Fang, C., & Wang, L. (2012). An effective shuffled frog-leaping algorithm for resource-constrained project scheduling problem. *Computers & Operations Research*, *39*(5), 890-901.

Fleszar, K., & Hindi, K.S. (2004). Solving the resource-constrained project scheduling problem by a variable neighbourhood search. *European Journal of Operational Research*, *155*(2), 402-413.

Glover, F. (1977). Heuristics for integer programming using surrogate constraints. *Decision Sciences*, *8*(1), 156-166.

Glover, F., Kelly, J.P., & Laguna, M. (1995). Genetic Algorithms and Tabu search: hybrids for optimization. *Computers & Operations Research*, *22*(1), 111-134.

Gonçalves, J.F., Resende, M.G., & Mendes, J.J. (2011). A biased random-key genetic algorithm with forward-backward improvement for the resource constrained project scheduling problem. *Journal of Heuristics*, *17*(5), 467-486.

Hartmann, S., & Kolisch, R. (2000). Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem. *European Journal of Operational Research*, *127*, 394-407.

Hartmann, S., & Briskorn, D. (2010). A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*, *207*(1), 1-14.

Herroelen, W. (2005). Project scheduling—Theory and practice. *Production and Operations Management*, *14*(4), 413-432.

Herroelen, W., De Reyck, B., & Demeulemeester, E. (1998). Resource-constrained project scheduling: a survey of recent developments. *Computers & Operations Research*, *25*(4), 279-302.

Hindi, K.S., Yang, H., & Fleszar, K. (2002). An evolutionary algorithm for resource-constrained project scheduling. *IEEE Transactions on Evolutionary Computation, 6*(5), 512-518.

Icmeli-Tukel, O., & Rom, W.O. (1997). Ensuring quality in resource constrained project scheduling. *European Journal of Operational Research* 103(3) 483-496.

Kochetov, Y., & Stolyar, A. (2003). *Evolutionary local search with variable neighborhood for the resource constrained project scheduling problem*. In Proceedings of the 3rd International Workshop on Computer Science and Information Technologies CSIT 2003, Ufa, Russia.

Kolisch, R. (1996). Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. *European Journal of Operational Research, 90,* 320-333.

Kolisch, R., & Hartmann, S. (1999). Heuristic algorithms for solving the resource-constrained project scheduling problem: Classification and computational analysis. In: J. Weglarz (Ed.), *Project Scheduling: Recent Models, Algorithms and Applications* (p. 147-178). Berlin, Germany: Kluwer Academic Publishers.

Kolisch, R., & Hartmann, S. (2006). Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European Journal of Operational Research*, *174*, 23-37.

Kolisch, R., & Padman, R. (2001). An integrated survey of deterministic project scheduling. *Omega*, *29*(3), 249-272.

Kolisch, R., Sprecher, A., & Drexl, A. (1992). *Characterization and generation of a general class of resource-constrained project scheduling problems - Easy and hard instances* (Research Report 301). Kiel, Germany: Instituten für Betriebswirtschaftslehre, Christian-Albrechts-Universität zu Kiel.

Kolisch, R., Sprecher, A., & Drexl, A. (1995). Characterization and generation of a general class of resource-constrained project scheduling problems. *Management Science, 41,* 1693-1703.

Kolisch, R., & Sprecher, A. (1996). PSPLIB - A project scheduling problem library. *European Journal of Operational Research*, *96*, 205–216.

Koulinas, G., Kotsikas, L., & Anagnostopoulos, K. (2014) A particle swarm optimization based hyper-heuristic algorithm for the classic resource constrained project scheduling problem. *Information Sciences*, *277*(1), 680-693.

Li, R.K.Y., & Willis J. (1992). An iterative scheduling technique for resource-constrained project scheduling. *European Journal of Operational Research*, *56*(3), 370-379.

Liberatore, M.J., & Pollack-Johnson, B. (2003). Factors influencing the usage and selection of project management software. *IEEE Transactions on Engineering Management* 50(2) 164-174.

Liberatore, M.J., Pollack-Johnson, B., & Smith, C.A. (2001). Project management in construction: Software use and research directions. *Journal of Construction Engineering and Management* 127(2) 101-107.

Lim, A., Ma, H., Rodrigues, B., Tan, S.T., & Xiao, F. (2013). New meta-heuristics for the resource-constrained project scheduling problem. *Flexible Services and Manufacturing Journal*, *25*(1-2), 48-73.

Marti, R., Laguna, M., & Glover, F. (2006). Principle of scatter search. *European Journal of Operational Research*, *169*(2), 359-372.

Mendes, J.J.M., Gonçalves, J.F., & Resende, M.G.C. (2009). A random key based genetic algorithm for the resource constrained project scheduling problem. *Computers & Operations Research, 36*(1), 92-109.

Mobini, M., Rabbani, M., Amalnik, M.S., Razmi, J., & Rahimi-Vahed, A.R. (2009). Using an enhanced scatter search algorithm for a resource-constrained project scheduling problem. *Soft Computing*, *13*(6), 597-610.

Mobini, M., Mobini, Z., & Rabbani, M. (2011). An Artificial Immune Algorithm for the project scheduling problem under resource constraints. *Applied Soft Computing, 11*(2), 1975-1982.

Möhring, R. H., Schulz, A. S., Stork, F., & Uetz, M. (2003). Solving project scheduling problems by minimum cut computations. *Management Science*, *49*(3), 330-350.

Nouri, N., Krichen, S., Ladhari, T., & Fatimah, P. (2013). *A discrete artificial bee colony algorithm for resource-constrained project scheduling problem*. In Proceedings of the 5th International Conference on Modeling, Simulation and Applied Optimization ICMSAO 2013, Hammamet, Tunisia (p. 1-6).

Özdamar, L., & Ulusoy, G. (1995). A survey on the resource-constrained project scheduling problem. *IIE transactions*, *27*(5), 574-586.

Palpant, M., Artigues, C., & Michelon, P. (2004). LSSPER: solving the resource-constrained project scheduling problem with large neighbourhood search. *Annals of Operations Research*, *131*(1-4), 237-257.

Paraskevopoulos, D.C., Tarantilis, C.D., & Ioannou, G. (2012). Solving project scheduling problems with resource constraints via an event list-based evolutionary algorithm. *Expert Systems with Applications 39*(4), 3983–3994.

Patterson, J. (1984). A Comparison of Exact Approaches for Solving the Multiple Constrained Resource Project Scheduling Problem. *Management Science, 30*(7), 854–867.

Proon, S., & Jin, M. (2011). A genetic algorithm with neighborhood search for the resource-constrained project scheduling problem. *Naval Research Logistics*, *58*(2), 73-82.

Ranjbar, M. (2008). Solving the resource-constrained project scheduling problem using filter-and-fan approach. *Applied mathematics and computation*, *201*(1), 313-318.

Ranjbar, M., De Reyck, B., & Kianfar, F. (2009). A hybrid scatter search for the discrete time/resource trade-off problem in project scheduling. *European Journal of Operational Research, 193*(1)*,* 39-48*.*

Resende, G.C., Ribeiro, C.C., Glover, F., & Marti, R. (2010) Scatter search and path-relinking: fundamentals, advances, and applications. In: M. Gendreau, and J.Y. Potvin (Eds.), *Handbook of Metaheuristics* (2nd ed., p. 87-107)*.* New York*,* NY, USA: Springer.

Sadeghi, A., Kalanaki, A., Noktehdan, A., Samghabadi, A. S., & Barzinpour, F. (2011). Using bees algorithm to solve the resource constrained project scheduling problem in PSPLIB. In: Q. Zhou (Ed.), *Theoretical and Mathematical Foundations of Computer Science, Communications in Computer and Information Science 164* (p. 486-494). Berlin Heidelberg, Germany: Springer.

Sprecher, A. (2002). Network decomposition techniques for resource-constrained project scheduling. *Journal of the Operational Research Society*, *53*(4), 405-414.

Tavares, L.V. (2002). A review of the contribution of operational research to project management. *European Journal of Operational Research*, *136*(1), 1-18.

Thammano, A., & Phu-Ang, A. (2012). A hybrid evolutionary algorithm for the resource-constrained project scheduling problem. *Artificial Life and Robotics*, *17*(2), 312-316.

Tormos, P., & Lova, A. (2001). A competitive heuristic solution technique for resource-constrained project scheduling. *Annals of Operations Research*, *102*, 65-81.

Valls, V., Quintanilla, S., & Ballestin, F. (2003). Resource-constrained project scheduling: A critical reordering heuristic. *European Journal of Operational Research*, *149*, 282–301.

Valls, V., Ballestin, F., & Quintanilla, S. (2004). A population-based approach to the resource-constrained project scheduling problem. *Annals of Operations Research*, *131*, 305–324.

Valls, V., Ballestin, F., & Quintanilla, S. (2005). Justification and RCPSP: A technique that pays. *European Journal of Operational Research*, *165*, 375-386.

Valls, V., Ballestin, F., & Quintanilla, S. (2008). A hybrid genetic algorithm for the resource-constrained project scheduling problem. *European Journal of Operational Research*, *185*(2), 495-508.

Wang, H., Li, T., & Lin, T. (2010). Efficient genetic algorithm for resource-constrained project scheduling problem. *Transactions of Tianjin University, 16*, 376-382.

Zamani, R. (2010a). An accelerating two-layer anchor search with application to the resource-constrained project scheduling problem. *IEEE Transactions on Evolutionary Computation*, *14*(6), 975-984.

Zamani, R. (2010b). A parallel complete anytime procedure for project scheduling under multiple resource constraints. *The International Journal of Advanced Manufacturing Technology*, *50*(1-4), 353-362.

Zamani, R. (2012). A polarized adaptive schedule generation scheme for the resource-constrained project scheduling problem. *RAIRO-Operations Research*, *46*(01), 23-39.

Zamani, R. (2013). A competitive magnet-based genetic algorithm for solving the resource-constrained project scheduling problem. *European Journal of Operational Research, 229*(2), 552-559.