TOP (2018) 26:283–308
https://doi.org/10.1007/s11750-018-0472-9

ORIGINAL PAPER

A biased random-key genetic algorithm for the project scheduling problem with flexible resources

Bernardo F. Almeida[1] · **Isabel Correia**[2] · **Francisco Saldanha-da-Gama**[1]

Received: 2 May 2017 / Accepted: 21 March 2018 / Published online: 16 April 2018

Abstract In this paper, we investigate a resource-constrained project scheduling problem with flexible resources. This is an $\mathcal{NP}$-hard combinatorial optimization problem that consists of scheduling a set of activities requiring specific resource units of several skills. The goal is to minimize the makespan of the project. We propose a biased random-key genetic algorithm for computing feasible solutions for the referred problem. We study different decoding mechanisms: an already existing method in the literature, a new adapted serial scheduling generation scheme, and a combination of both. The new procedure is tested using a set of benchmark instances of the problem. The results provide strong evidence that the new heuristic is robust and yields high-quality feasible solutions.

Keywords Resource-constrained project scheduling · Flexible resources · Biased random-key genetic algorithm

Mathematics Subject Classification 90B35 (Scheduling theory, deterministic) · 90C59 (Approximation methods and heuristics)

✉ Bernardo F. Almeida
bernardo.almeida@alunos.fc.ul.pt

[1] Departamento de Estatística e Investigação Operacional/Centro de Matemática, Aplicações Fundamentais e Investigação Operacional, Faculdade de Ciências, Universidade de Lisboa, 1749-016 Lisbon, Portugal

[2] Departamento de Matemática/Centro de Matemática e Aplicações, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, 2829-516 Caparica, Portugal

1 Introduction

The resource-constrained project scheduling problem (RCPSP) is a well-known problem that consists of scheduling a set of activities, each of which requiring specific resources that are available in limited quantities. The goal is to minimize the makespan. This problem has been widely studied in the literature along with many variants. For an overview on this topic, the reader can refer to Herroelen and Demeulemeester (1998), Brucker et al. (1999), Hartmann and Briskorn (2010), and Węglarz et al. (2011), as well as to the references therein.

In many RCPSPs, resources are flexible, which means that each resource masters more than one skill. This is particularly true when human resources or multi-purpose machines are involved in a project. Typically, in this context, activities require specific resource units of several skills to be processed. The assignment of a resource to an activity comprehends the decision of the skill it will perform, chosen among the skills required by that activity and mastered by this resource. This extension of the RCPSP defines the basic setting of the so-called resource-constrained project scheduling problem with flexible resources. Hereafter, we will shorten this name and refer to the problem as the project scheduling problem with flexible resources (PSPFR). This problem and some of its variants have been introduced and studied in the literature as we detail next. Li and Womer (2009) develop a hybrid Benders decomposition for a PSPFR whose objective function regards the minimization of the total cost associated with the resources, while assuring that a predefined deadline for the project is satisfied. In such problem, each activity requires only one resource unit per each skill needed for its execution. Correia et al. (2012) propose a mixed-integer linear programming formulation for a PSPFR where the activities may require several resources per each skill needed for their execution. The objective is to minimize the makespan of the project. Several sets of additional inequalities and reduction tests are also proposed. The impact of using fixed and variable costs in a PSPFR is studied by Correia and Saldanha-da-Gama (2014) who develop a mixed-integer model with a non-linear objective function. The model is linearized and is strengthened through the inclusion of additional inequalities. Finally, the enhanced formulation is tested using an off-the-shelf solver. More recently, Correia and Saldanha-da-Gama (2015) propose a general modeling framework for a PSPFR. The authors discuss several modeling issues and propose several procedures for enhancing the models.

Flexible resources have also been considered in the context of project portfolio problems, but, typically, with additional assumptions. For example, in the works by Gutjahr et al. (2008) and Heimerl and Kolisch (2010), no sequencing decisions involving the activities have to be made. The problem studied by Gutjahr et al. (2008) involves the selection of the projects to be executed among a set of available projects. The activities of each project must be executed within specific and predetermined time-windows. Each activity requires resources, each of which mastering several skills at different levels of efficiency that may change over time. Heimerl and Kolisch (2010) address a project scheduling and staffing problem with hierarchical levels of skills where the goal is to minimize the variable costs associated with the resources. A related staffing problem, which also considers that the skills can be performed at different levels of efficiency, was investigated by Walter and Zimmermann (2016).

The problem investigated in the current paper is the PSPFR studied by Correia et al. (2012). The methodology proposed in that paper turned out to be effective only for small-sized instances. Almeida et al. (2016) proposed a heuristic for tackling larger instances of such problem by extending the well-known parallel scheduling scheme (Kolisch 1996a, b). Despite the significant advances represented by the work published by Almeida et al. (2016), much work still remains to be done namely in terms of improving the quality of the feasible solutions obtained. The current paper emerges in this context. In particular, we propose a constructive heuristic for the PSPFR based on the serial scheduling scheme as well as a biased random-key genetic algorithm (BRKGA) aiming at coordinating that scheme with the parallel scheduling scheme of Almeida et al. (2016). To the best of the authors' knowledge, a BRKGA has never been proposed for a PSPFR, although it has been applied successfully to RCPSPs (see Gonçalves et al. 2008; Mendes et al. 2009, and Gonçalves et al. 2011). BRKGA algorithms have also been successfully applied to other optimization problems such as packing (Gonçalves and Resende 2013), facility layout (Gonçalves and Resende 2015), capacitated minimum spanning trees (Ruiz et al. 2015), among others.

We note that the need for the development of heuristics for the PSPFR has also been observed for the RCPSP as attested, for instance, in the works by Hartmann and Kolisch (2000), Kolisch and Hartmann (1999), and Kolisch and Hartmann (2006), who provide valuable contributions in this context.

The remainder of this paper is organized as follows. In Sect. 2, we describe the PSPFR and some of its properties and additional concepts. Sect. 3 is dedicated to the heuristic we propose. Finally, in Sect. 4, we report the computational tests performed to evaluate the BRKGA. The paper ends with an overview of the work done and with some directions for further research.

2 Problem description and properties

As we have already mentioned, the problem studied in this work is the PSPFR studied by Correia et al. (2012). To make the paper self-contained, we describe next the problem. In addition, we introduce some relevant concepts for the remainder of the paper.

Consider a project represented by an activity-on-node network $G = (V, E)$ where $V = \{0, 1, \ldots, j, \ldots, n+1\}$ denotes the set of activities and E is the set of arcs representing the precedence relations between the activities. Activities 0 and $n+1$ are dummy activities that represent the start and the conclusion of the project, respectively. An arc (i, j) is in E if activity i is a direct predecessor of activity j. For every $j \in V \backslash \{0\}$, we denote by $Pred(j)$ the set of predecessors of j; for $j \in V \backslash \{n+1\}$, $Succ(j)$ is the set of its successors. The weight of an arc (i, j) is denoted by p_i and represents the processing time of activity i. Preemption is not allowed, i.e., once an activity starts being executed, it cannot be interrupted.

A set of renewable resources $\mathcal{R} = \{1, \ldots, k, \ldots, K\}$ mastering one or several skills is required to execute the activities. The pool of available skills is denoted by $\mathcal{L} = \{1, \ldots, l, \ldots, L\}$. The set of skills mastered by resource $k \in \mathcal{R}$ is denoted by $\mathcal{L}^k$, and the set of skills required by an activity $j \in V$ is represented by $\mathcal{L}_j$. The

number of resources mastering skill $l \in \mathcal{L}_j$ required by activity $j \in V$ is denoted by r_{jl}. A resource can be involved in only one activity at a time and for a single skill; furthermore, once it is assigned to an activity with a skill, it remains so during the whole processing time of this activity.

The goal of the PSPFR is to determine for each activity $j \in V \backslash \{0, n+1\}$, (i) its starting time, hereafter denoted by S_j and (ii) the pairs (resource, skill) that should be assigned to it. The objective is to minimize the project's makespan.

In the PSPFR, it is also assumed that the values p_j and r_{jl} ($j \in V \backslash \{0, n+1\}; l \in \mathcal{L}_j$) are positive integers and are zero for the dummy activities. As a result, the makespan of the project is a positive integer less than or equal to $\sum_{j \in V} p_j$, which is the value obtained by a sequential execution of the activities.

2.1 Additional concepts and properties

The upper bound given by $\sum_{j \in V} p_j$ can usually be improved when some activities can be executed in parallel, i.e., can have their execution overlapping for some time. However, in the PSPFR, it is not trivial to check whether two or more activities can be executed in parallel. The difficulty emerges from the fact that we have (multi-skill) resource constraints in addition to the usual precedence ones. This aspect may be crucial when it comes to obtaining good feasible solutions for the problem. Above all, it is important to have a mechanism, as efficient as possible, to check whether two or more activities can overlap in time. We discuss this issue next.

Assume that, at a given time $t > 0$, some activities have already been scheduled to have their execution time starting before t. Denote by UV the activities still to be scheduled and by W_t a subset of UV containing only activities (not necessarily all), such that all their predecessors are already completed at time t (or that have no predecessors). Can we set to t the starting time of all the activities in W_t?

This query has a positive answer if all the resources available at time t can meet the skill requirements of all the activities in W_t at the same time. In that case, the literature on PSPFR denominates W_t as a set of compatible activities (see Correia et al. 2012). This concept is equivalent to the notion of feasible set proposed by Mingozzi et al. (1998) for the RCPSP. Checking the existence of precedence relations between each pair of activities can be done straightforwardly. However, this is not the case when we need to verify whether there are enough resources to meet all the skill requirements of the activities in W_t simultaneously.

Let us denote by $\mathcal{Z}_{W_t}$ the set of resources that are available at time t and which master at least one skill required by at least one activity in W_t. In addition, denote by $\mathcal{L}_{W_t}$ the set of skills required to process the activities in W_t. Checking whether the set of activities W_t is compatible can be done in polynomial time by solving a flow feasibility problem in an appropriate network that we denote by $G_{W_t} = (V_{W_t}, E_{W_t})$, which is built as follows:

- The set of nodes V_{W_t} contains: (i) a source node, v_0; (ii) a set of nodes—one for each resource in $\mathcal{Z}_{W_t}$; (iii) a set of nodes—one for each skill in $\mathcal{L}_{W_t}$; (iv) a sink node, v_s.

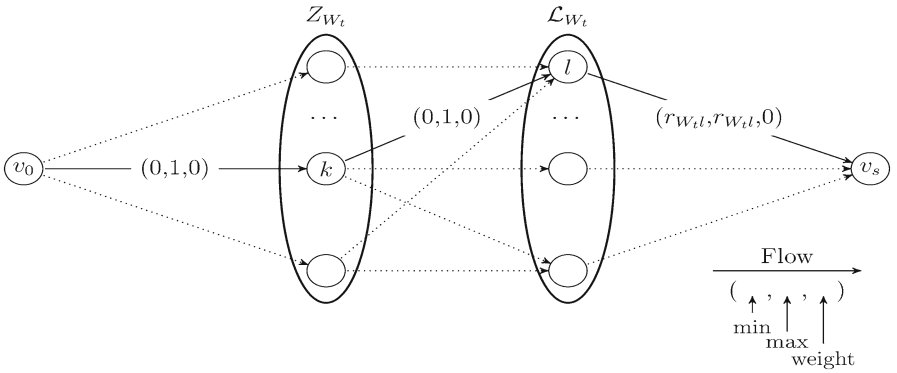

Fig. 1 Illustration of the auxiliary graph G_{W_t}

- The set of arcs E_{W_t} is defined by: (i) a set of arcs (v_0, k), $k \in Z_{W_t}$ with minimum throughput 0, capacity 1 and unitary cost 0; (ii) a set of arcs (k, l), $l \in (\mathcal{L}^k \cap \mathcal{L}_{W_t})$ with minimum throughput 0, capacity 1 and unitary cost 0; (iii) a set of arcs (l, v_s), $l \in \mathcal{L}_{W_t}$ with minimum throughput and capacity given by $r_{W_t l}$ and unitary cost equal to 0, where $r_{W_t l}$ denotes the number of resource units required to process skill l for all the activities in W_t.

The network G_{W_t} is illustrated in Fig. 1. If a feasible flow exists in this auxiliary network, then we know that there are enough resources to start processing all the activities in W_t, at time t. However, as we explain next, we can go deeper in this analysis, which requires revisiting some additional concepts.

2.1.1 Resource weights

A feasible flow in G_{W_t} induces an assignment of the resources in $\mathcal{Z}_{W_t}$ to the skills required by the activities in W_t. Such a flow may not be unique due to the flexible nature of the resources, which may render different possibilities of meeting the skill demands of W_t by varying (i) the selected resources from the set $\mathcal{Z}_{W_t}$ or (ii) the skill $l \in \mathcal{L}_{W_t}$ that each resource in (i) is assigned to perform, or both (i) and (ii).

Since each resource k masters a specific set of skills, $\mathcal{L}^k$, we may characterize a resource as being more versatile than others (e.g., by mastering more skills), more important (e.g., by mastering scarce or highly required skills), etc. Hence, to compute feasible solutions, it becomes relevant to determine the best resource to meet each unitary skill demand, because this assignment may have impact in future iterations and thus compromise the quality of the derived schedule. In Almeida et al. (2016), this fact motivated the development of a new concept—the *weight of a resource*: for some resource $k \in \mathcal{R}$, its *weight* is denoted by w_k and represents a measure resulting from selecting that resource to execute a skill mastered by it and required by at least one activity $j \in W_t$.

2.1.2 Resource assignment

Almeida et al. (2016) propose assigning the resources to the skills required by W_t by solving a min-cost flow problem in a modified network $\widetilde{G}_{W_t} = (V_{W_t}, E_{W_t})$ obtained from $G_{W_t} = (V_{W_t}, E_{W_t})$ by replacing the weight of each arc (v_0, k), $k \in \mathcal{Z}_{W_t}$, by the corresponding resource weight, w_k. That problem will be named $MCNFP(\widetilde{G}_{W_t})$.

Again, a feasible flow in $\widetilde{G}_{W_t}$ indicates that W_t is a set of compatible activities. However, an optimal solution to $MCNFP(\widetilde{G}_{W_t})$ only provides an assignment of the resources $k \in \mathcal{Z}_{W_t}$ to the skills $l \in (\mathcal{L}^k \cap \mathcal{L}_{W_t})$; it does not indicate the activity each resource is allocated to. In particular, for every $l \in \mathcal{L}_{W_t}$, we obtain a set of resources $\mathcal{X}_{W_t l} \subseteq \mathcal{Z}_{W_t}$ that meet the skill requirements of activity l. In Almeida et al. (2016), a heuristic procedure is proposed for assigning the resources with larger weights to the activities with smaller processing times, in an attempt to make those resources (looked as valuable) free as soon as possible and thus available to be assigned to other activities that may need them.

2.1.3 Activity priorities

Likewise for the resources, when we look deeply into the activities, we realize that a sort of ranking can be devised. In an attempt to use some rational mechanism for building that rank, Almeida et al. (2016) computed a priority value, pv_j, for each activity $j \in V \backslash \{0, n+1\}$ using the well-known activity priority rules already proposed for the RCPSP (Kolisch 1996a; Demeulemeester and Herroelen 2002). In the next section, we show that the priority values for the activities can be dynamically adapted.

3 A new constructive heuristic

In this section, we detail the new heuristic proposed in this work for the PSPFR. We start by reviewing basic aspects related with the underlying metaheuristic, and then, we introduce the specifications for our problem.

3.1 BRKGAs

In a BRKGA, a population of chromosomes evolves over a number of generations until the defined stopping criteria are met (cf. Gonçalves and Resende 2011). Each chromosome encodes a solution of the problem and is represented by an m-dimensional vector of real numbers in the interval [0,1]—the random keys. The BRKGA differs from the classical genetic algorithm in the following aspects: (i) the chromosomes with the best fitness values in one generation (elite population) are copied unchanged to the next generation; (ii) in each generation, a new set of chromosomes is generated from scratch and is included in the population—the mutants, instead of using the classical mutation operators; (iii) a parameterized crossover occurs between a parent selected from the set of elite chromosomes and a parent select from the set of non-elite chromosomes. Algorithm 1 depicts a generic BRKGA where p denotes the number of chromosomes in the population; m is the number of genes in each chromosome;

p_e and p_m denote the percentage of elite and mutant chromosomes in the population, respectively; ρ_e represents the probability of a descendant inheriting an allele from its elite parent; $U[0, 1]$ denotes a random number in the interval $[0, 1]$; g is a generation counter; and finally, c^* and f^* denote, respectively, the best chromosome and its fitness value.

Algorithm 1: A Biased Random-Key Genetic Algorithm (BRKGA)

```
   Data: p, p_e, p_m, m, ρ_e
   Result: c*, f*
1  begin
2      Generate initial population P_1 with p chromosomes where each allele is U[0, 1];
3      Compute the fitness of the p chromosomes using the decoder;
4      Initialize f* and c*;
5      g ⟵ 1;
6      while the stoping criteria are not satisfied do
7          Save in the set P_g^e the ⌈p_e × p⌉ most fit chromosomes of P_g ;
8          Copy P_g^e into P_{g+1};
9          Generate ⌈p_m × p⌉ mutants, compute their fitness and add them to P_{g+1};
10         for i = 1 to (p − ⌈p_e × p⌉ − ⌈p_m × p⌉) do
11             Randomly select parent c_1 from P_g^e and parent c_2 from P_g \ P_g^e;
12             for j = 1 to m do
13                 u = U[0, 1];
14                 if u < ρ_e then
15                     c_3[j] ⟵ c_1[j];
16                 else
17                     c_3[j] ⟵ c_2[j];
18                 end
19             end
20             Compute the fitness of c_3, using the decoder, and copy c_3 to P_{g+1};
21         end
22         g ⟵ g + 1;
23         if a better chromosome was found then
24             Update f* and c*;
25         end
26     end
27 end
```

3.2 Application to the PSPFR

In this section, we introduce the structure of the chromosomes and the decoder that we propose for the PSPFR. These are the components of Algorithm 1 that are specific to each particular problem.

3.2.1 Decoder

Since the PSPFR is an extension of the RCPSP, two natural decoders for the former emerge from extending the well-known parallel and serial scheduling schemes existing for the latter designated by PSS and SSS, respectively (cf. Kolisch 1996b).

Although the SSS and the PSS can be looked at as simple heuristic strategies for the RCPSP, their extension to the PSPFR is more complex due to the selection and assignment of the flexible resources to the activities of the project.

The PSS was extended to the PSPFR by Almeida et al. (2016). This is an iterative method where, initially, the time counter is set to 0. In each iteration, which is associated with a specific moment in time t, the set of precedence feasible activities W_t is built. If W_t is empty, the value of t is incremented to the next time in which some activity becomes available for execution due to the completion of all its predecessors. Otherwise, either there are enough resources—among those available at time t—to meet all the skill requirements of the activities in W_t, and hence, all these activities are set to start being processed at time t, or this is not the case; and thus, the activity with the smallest priority value is successively removed from W_t until the resulting set is either empty or all the skill requirements of its activities can be met. The reader should refer to Almeida et al. (2016) for all the details.

Concerning the SSS, we propose next its adaptation to the PSPFR. Again, it consists of an iterative procedure that starts by setting the time counter t to 0. In each iteration, the activity j^* with the highest priority value and whose predecessors have already been executed is selected to be scheduled. The time counter t is then set to the maximum completion time across all the predecessors of j^*. Activity j^* is scheduled to start at time t if the resources available from time t to time $t + p_{j^*} - 1$ are enough to fulfil all its skill requirements; otherwise, t, is moved forward to the next completion time of the activities already scheduled. Again, the availability of resources is checked. This process repeats until eventually activity j^* is scheduled. In the SSS that we are proposing, the set W_t contains only the activity selected to be scheduled next. This means that, in this case, we no longer need to consider this set. Nevertheless, we keep it for exposition purposes, because this notation was also used in Almeida et al. (2016). The new scheme which we are proposing is fully detailed in Algorithm 2.

In contrast to the PSS, in the SSS which we are proposing, it is possible to have already scheduled activities which start after time t. In fact, when an activity, say j^*, is selected to be scheduled, the time t is moved to the maximum completion time of all predecessors of j^*. Therefore, it is possible to have other already scheduled activities which have the necessary resources allocated and their starting times higher than t. Hence, for a given time t, deciding whether activity j^* can start being processed at that time, requires checking if the resources available in every time slot where j^* will be in progress, from t to its provisional finish time $t + p_{j^*} - 1$, can fulfil all its skill requirements.

The PSS by Almeida et al. (2016) and the SSS just presented can also be applied to the problem obtained by reversing all the arcs in the precedence network. This problem is equivalent to planning the project backwards starting at the end and moving regressively to the beginning. Naturally, the problem to be solved is the same, but the order by which each activity is scheduled may be different, and hence, a different resource selection and assignment may occur, which results in a schedule with, possibly, a different makespan. This concept of backward planning has already been applied to the RCPSP by Li and Willis (1992), Özdamar (1999), Klein (2000), and Alcaraz and Maroto (2001), to mention a few.

Algorithm 2: A Serial Scheduling Scheme (SSS) for the PSPFR (Decoder)

Data: $V, E, Pred(j), Succ(j), \mathcal{R}, \mathcal{L}, \mathcal{L}_j, \mathcal{L}^k, p_j, r_{jl}, pv_j, w_k : j \in V, k \in \mathcal{R}, l \in \mathcal{L}_j$
Result: *makespan*
begin
  $UV \longleftarrow V \backslash \{0, n+1\}$; $t \longleftarrow 0$; $S_0 \longleftarrow 0$; $S_j \longleftarrow \infty$, $j \in UV$;
  while $UV \neq \emptyset$ **do**
    Find $j^* : pv_{j^*} = \max\{pv_j : j \in UV \wedge Pred(j) \cap UV = \emptyset\}$;
    if $Pred(j^*) = \emptyset$ **then**
      $t \longleftarrow 0$;
    else
      $t \longleftarrow \max\{S_i + p_i : i \in Pred(j^*)\}$;
    end
    Compute $\mathcal{R}_{j^*} = \{k \in \mathcal{R} : \mathcal{L}^k \cap \mathcal{L}_{j^*} \neq \emptyset\}$ // resources with skills required by activity j^*;
    $W_t \longleftarrow \{j^*\}$;
    while j^* *unscheduled* **do**
      $\mathcal{Z}_{W_t} \longleftarrow \emptyset$;
      for $k \in \mathcal{R}_{j^*}$ **do**
        if k *is available in every time instant* $\{t, ..., t + p_{j^*} - 1\}$ **then**
          $\mathcal{Z}_{W_t} \longleftarrow \mathcal{Z}_{W_t} \cup \{k\}$;
        end
      end
      Solve the $MCNFP(\widetilde{G}_{W_t})$;
      if $MCNFP(\widetilde{G}_{W_t})$ *has a feasible solution* **then**
        $S_{j^*} \longleftarrow t$;
        For each skill $l \in \mathcal{L}_{j^*}$, assign the resources $k \in \mathcal{X}_{W_t l}$ and set them busy within $t \in \{S_{j^*}, \ldots, S_{j^*} + p_{j^*} - 1\}$;
        $UV \longleftarrow UV \backslash \{j^*\}$;
      else
        $t \longleftarrow \min\{S_u + p_u : u \notin UV \wedge S_u + p_u > t\}$ // increment t;
      end
    end
  end
  $makespan \longleftarrow \max\{S_j + p_j : j \in V\}$;
end

3.2.2 Chromosomes' structure

The structure of the chromosomes that we propose is inspired on the chromosomes considered by Alcaraz and Maroto (2001, 2006), Hartmann (2002), Gonçalves et al. (2008), Mendes et al. (2009), and Gonçalves et al. (2011). We define a chromosome as having $m = n + K + 2$ genes, where n is the number of non-dummy activities, K is the number of resources, and the last two positions are associated with the decoder. In particular, one of the last two genes indicates the scheduling generation scheme (PSS or SSS), while the other refers to the precedence network scheme (original or reversed). Figure 2 illustrates this chromosome structure. A chromosome with a gene indicating whether a PSS or a SSS shall be used was proposed by Hartmann (2002), while Alcaraz and Maroto (2001) consider a chromosome with an additional gene that indicates whether the original or reversed precedence network is used. In Alcaraz and Maroto (2006), both genes were included in a chromosome structure which they

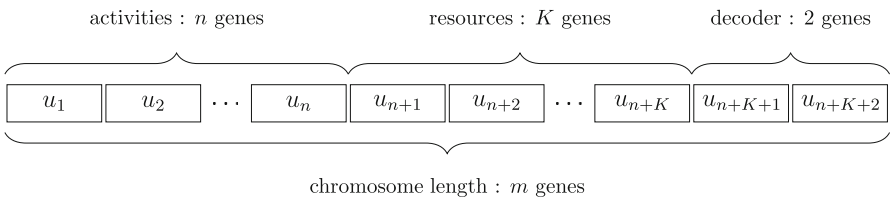

Fig. 2 Generic chromosome for the developed BRKGA

Table 1 Decoder parameters

Gene number	Parameter	$0 \leq u_i < 0.5$	$0.5 \leq u_i \leq 1$
$n + K + 1$	Scheduling scheme	SSS	PSS
$n + K + 2$	Precedence network	Original	Reversed

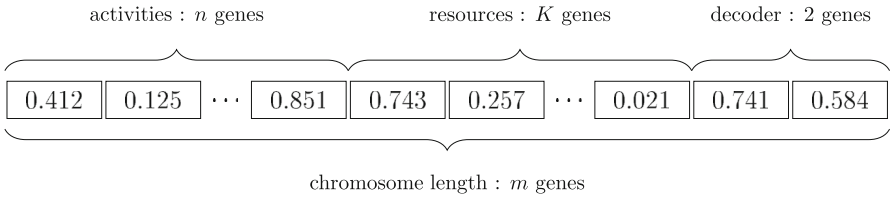

Fig. 3 Example of a chromosome

propose for the RCPSP. To the best of the authors' knowledge, a chromosome that includes genes associated with the resources along with a gene for decoder selection and a gene for encoding the precedence network scheme has never been attempted before.

The PSS and SSS for the PSPFR require a priority value, pv_j, to each activity $j \in V\backslash\{0, n+1\}$ and a weight value, w_k, to each resource $k \in \mathcal{R}$. This information will be embedded in the chromosome structure which we propose where each allele is a random uniform number in [0,1]. In particular, the value of the first n genes will give the priority value of the corresponding activity, while the value of the next K genes will give the weights of the resources. The information associated with the last two genes is provided in Table 1.

Figure 3 illustrates a chromosome within our BRKGA that will be decoded with the PSS applied to the reverse precedence network.

Using this chromosome structure, the same sequence of activities' priority values and resources' weight values may originate four different feasible solutions for the PSPFR, by changing the values of the last two genes. This kind of structure may help diversifying the exploration of the solution space. Another positive feature of this representation is its versatility. In fact, if the values of the last two genes are fixed for all the chromosomes generated in the BRKGA, this means that all chromosomes will be decoded with the same algorithm.

4 Computational experience

In this section, we report the numerical experiments performed to evaluate the performance of the BRKGA. The algorithm and the decoders were coded in C++. The min-cost flow problems were solved by integrating IBM ILOG CPLEX 12.6 with C++ through Concert Technology. All computational experiments were performed on a machine running an Intel Core i7 4770K with 32 GB of RAM.

This section is organized as follows. In Sect. 4.1, we present the benchmark instances used in the tests. In Sect. 4.2, we discuss the fine-tuning of the algorithm. Finally, in Sect. 4.3, the computational results are reported.

4.1 Test instances

Two different sets of instances, hereafter denoted by *Set 1* and *Set 2*, were considered. The *Set 1* contains the 216 instances generated by Correia et al. (2012), whereas *Set 2* refers to 216 larger instances built using the generator proposed by Almeida et al. (2015), which were already used by Almeida et al. (2016). The reader should refer to those works for all the details regarding the instances used.

From the work by Correia et al. (2012), we know the optimal value for 203 instances of *Set 1* and a lower bound for the remaining 13 instances in this set.

Apart from other parameters, the instances were generated with different values of network complexity (NC), skill factor (SF), and modified resource strength (MRS).

4.2 Fine-tuning the BRKGA

By making use of a set of preliminary tests, it was possible to find good values for the parameters of the BRKGA. Nevertheless, to get a hint in terms of good ranges for those parameters, we referred again to Gonçalves et al. (2008), Mendes et al. (2009), and Gonçalves et al. (2011). The preliminary experiments were performed on instances from *Set 1*, since we know the optimal value for most of them. We start by considering three variants of the BRKGA with regard to the type of decoder employed, namely: (i) all the chromosomes are decoded with the PSS; (ii) all the chromosomes are decoded with the SSS; and (iii) the decoder applied to each chromosome depends on the value of the allele in the position $n + K + 1$. These experiments allow us to evaluate if one of the variants performs better than the others. Regarding the precedence network schemes (original or reversed), the computational results provided by Almeida et al. (2016) indicate no dominance. Moreover it was not even possible to identify any class of instances (defined by SF, NC, and MRS) where a precedence network scheme performs better than the other. Therefore, we decided to use both network schemes to diversify the search space and this means that, in each chromosome, the gene $n+K+2$ indicates the scheme used by the decoder.

Regarding the other BRKGA parameters, Table 2 presents the values considered whose combination originates 36 distinct configurations. Each BRKGA configuration was set to run 5 times for each instance (using a distinct seed for the random number generation at the beginning of each run). Accordingly, for each instance tested, it is

Table 2 Preliminary tests—range of values for the BRKGA parameters

Population size (p)	$5 \times n$ Chromosomes
Probability of inheriting an allele from the elite parent (ρ_e)	0.7
Percentage of elite solutions in each generation (p_e)	{10, 15}
Percentage of mutant solutions in each generation (p_m)	{15, 20, 30}
Decoder (δ)*	{PSS, SSS, Both}
Stopping criterion ($\mathcal{G}$)	$n/2$ and $5 \times n$ generations
Fitness	Makespan (smaller is better)

* We considered both original and reversed precedence networks

possible to compute the average and minimum values for the makespan. The runs are independent from each other and terminate when the maximum number of generations is reached.

We present in Table 3 the overall results for every configuration tested. This table consists of two main sets of columns: (i) columns 4–8 refer to the results after $n/2$ generations and (ii) columns 9–13 to the results after $5 \times n$ generations. The scheduling generation scheme considered as well as the values of p_e and p_m are presented in columns 1–3, respectively. Each row depicts the average values for the 216 instances in *Set 1*. In terms of percentage gaps, their average and minimum values are indicated in columns 4 and 9, and columns 5 and 10, respectively. The average makespan values and their associated standard deviations are presented in columns 6 and 11, and columns 7 and 12, respectively. Columns 8 and 13 contain the CPU time (in seconds) used in the 5 runs until reaching $\frac{n}{2}$ generations and $5 \times n$ generations, respectively. Each gap (in percentage) is computed as $100 \times (Z^{\mathcal{B}} - Z^{LB})/Z^{LB}$, where $Z^{\mathcal{B}}$ denotes the upper bound provided by the BRKGA and Z^{LB} denotes the optimal value or the best known lower bound.

The makespan values and gaps presented in Table 3 allow us to conclude that, as expected, with $5 \times n$ generations, it was possible to obtain better upper bounds than those obtained with $n/2$ generations. However, this improvement in the quality of the feasible solutions was achieved at the expense of a CPU time one order of magnitude higher, which seems not to be compensatory. Hence, we narrow our analysis using the results obtained after $n/2$ generations.

Looking into such results (max. $n/2$ generations), we conclude that the use of the PSS in all chromosomes provided the best results in terms of the average percentage gaps and CPU time. However, the use of both decoders yielded the best minimum gaps for the majority of the combinations of p_e and p_m. Among the six combinations of p_e and p_m, we adopt $p_e = 10\%$ and $p_m = 30\%$, since this combination originated the best average and minimum gaps.

In Almeida et al. (2016), we observe that higher percentage gaps were obtained for instances having less resources. Since those instances were also the ones that required less computational time, we take advantage of this behavior and consider a population size determined not only by the number of activities (n) but also by the number of resources (K) in each instance. Once again, we used all the instances in *Set 1* and

Table 3 BRKGA—results for the preliminary tests

δ	p_e (%)	p_m (%)	$\mathcal{G} = \frac{n}{2}$ generations					$\mathcal{G} = 5 \times n$ generations				
			Gap (%)		Makespan		Total time (CPU sec.)	Gap (%)		Makespan		Total time (CPU sec.)
			Avg.	Min.	Avg.	Std.		Avg.	Min.	Avg.	Std.	
PSS	10	15	1.19	0.88	52.296	0.163	55.188	0.92	0.75	52.162	0.095	538.735
		20	1.19	0.83	52.298	0.196	55.322	0.87	0.70	52.135	0.112	540.412
		30	1.17	0.86	52.294	0.176	55.558	0.84	0.69	52.114	0.080	543.484
	15	15	1.20	0.87	52.300	0.179	52.668	0.89	0.73	52.141	0.098	510.533
		20	1.18	0.87	52.294	0.161	52.688	0.88	0.72	52.136	0.085	510.715
		30	1.25	0.92	52.331	0.179	52.785	0.87	0.72	52.130	0.085	513.483
SSS	10	15	1.71	1.08	52.531	0.301	76.939	1.21	0.83	52.281	0.191	750.451
		20	1.79	1.23	52.582	0.278	77.054	1.23	0.81	52.302	0.221	753.332
		30	1.82	1.28	52.604	0.274	77.079	1.20	0.85	52.284	0.195	754.394
	15	15	1.68	1.11	52.531	0.278	73.416	1.19	0.83	52.279	0.205	712.074
		20	1.82	1.16	52.600	0.296	73.436	1.16	0.79	52.262	0.193	712.910
		30	1.82	1.24	52.606	0.286	73.477	1.10	0.75	52.239	0.178	714.160
Both	10	15	1.19	0.82	52.298	0.191	63.326	0.92	0.73	52.158	0.109	613.704
		20	1.23	0.86	52.314	0.197	63.668	0.89	0.72	52.144	0.118	617.794
		30	1.21	0.78	52.310	0.225	64.405	0.88	0.74	52.140	0.086	627.068
	15	15	1.20	0.84	52.303	0.186	60.838	0.89	0.73	52.142	0.094	585.492
		20	1.23	0.87	52.317	0.191	60.640	0.86	0.71	52.131	0.097	583.218
		30	1.26	0.90	52.333	0.203	61.265	0.86	0.72	52.124	0.083	592.537

tested a population $p = 5 \times \lceil \frac{n \times n}{K} \rceil$, considering the adopted values of p_e and p_m, and decoding all chromosomes with the PSS. By fixing the number of activities (which is 20 for all these instances), such value for p originates larger population sizes for the instances with a smaller number of resources. The preliminary computations show that this p value yields an improvement in the average solution gaps (when compared to the population of size $p = 5 \times n$) of roughly 6%, at the expense of an 8% increase in the total time, for the BRKGA configuration having $p_e = 10\%$, $p_m = 30\%$ and only the PSS decoder.

Considering the information resulting from the preliminary computations, we decided to deepen our computational tests considering the following configurations: (i) $p = 5 \times \lceil \frac{n \times n}{K} \rceil$, $p_e = 10\%$, $p_m = 30\%$, $\rho_e = 0.7$, and $\delta = PSS$; (ii) $p = 5 \times \lceil \frac{n \times n}{K} \rceil$, $p_e = 10\%$, $p_m = 30\%$, $\rho_e = 0.7$, and $\delta = Both$. The maximum number of generations assumed as the stopping criterion is $n/2$.

4.3 Computational results

We discuss separately the results obtained for instances in Sets 1 and 2. In what follows, we denote the BRKGA with the single decoder PSS as $\text{BRKGA}_{\text{PSS}}$ and the BRKGA with the two decoders as $\text{BRKGA}_{\text{Both}}$.

4.3.1 Results for the instances in Set 1

In Table 4, we present the results for the instances in *Set 1* considering the 36 groups induced by the values of SF, NC, and MRS. Each row aggregates six instances. The gaps were computed as explained above. In Table 5, we summarize these results.

The information presented in Tables 4 and 5 is organized as follows: columns 1–3 indicate the characteristics of the instances; columns 4–10 contain the results obtained with the $\text{BRKGA}_{\text{PSS}}$. In particular, columns 4, 5, and 6 are associated with the average, minimum, and maximum gaps achieved; columns 7 and 8 depict the average makespan values and their associated standard deviation, respectively; and column 9 contains the total CPU time (seconds) required to perform the 5 runs; columns 10–14 contain the same information as before but for $\text{BRKGA}_{\text{Both}}$; finally, columns 15–16 contain the gaps obtained by the heuristic proposed in Almeida et al. (2016) and the corresponding CPU time.

From Table 4, we observe that the average, minimum, and maximum gaps achieved by the $\text{BRKGA}_{\text{PSS}}$ improve the results provided by Almeida et al. (2016) in 26, 27, and 23 out of the 36 classes of instances, respectively. In terms of average and minimum gaps, the $\text{BRKGA}_{\text{PSS}}$ achieved the same results as the heuristic of Almeida et al. (2016) in 8 classes of instances, 7 of which correspond to a gap of 0.0%. The 12 classes of instances whose minimum gaps obtained by the $\text{BRKGA}_{\text{PSS}}$ are highlighted in boldface indicate that the $\text{BRKGA}_{\text{PSS}}$ was able to find the optimal value of all these 72 instances, but the constructive heuristic of Almeida et al. (2016) was not.

The higher gaps yielded by the $\text{BRKGA}_{\text{PSS}}$ are associated with instances with SF = var., NC = 1.5, and MRS = 0.1667. The higher values of the standard deviation of the makespan occur more often in the sets of instances associated with the smallest

Table 4 *Set 1*: Comparative analysis—Detail

SF	NC	MRS	BRKGA 5 runs $p = 5 \times \lceil \frac{n \times n}{K} \rceil$, $p_e = 10\%$, $p_m = 30\%$, $\rho_e = 70\%$, $\mathcal{G} = \frac{n}{2}$												Almeida et al. (2016)	
			$\delta = PSS$						$\delta = Both$							
			Gap (%)			Makespan		Total time (CPU sec.)	Gap (%)			Makespan		Total time (CPU sec.)	Gap (%)	Total time (CPU sec.)
			Avg.	Min.	Max.	Avg.	Std.		Avg.	Min.	Max.	Avg.	Std.			
1	1.5	0.1250	6.33	6.13	6.81	52.433	0.149	61.027	6.47	5.83	7.78	52.500	0.448	66.355	9.47	1.096
		0.1563	2.89	2.89	2.89	49.167	0.000	55.743	2.97	2.89	3.28	49.200	0.075	63.376	4.37	1.145
		0.1875	0.00	0.00	0.00	48.667	0.000	56.852	0.00	0.00	0.00	48.667	0.000	67.529	0.00	1.374
	1.8	0.1250	0.73	**0.00**	1.28	53.033	0.289	59.165	0.77	**0.00**	1.28	53.067	0.322	64.993	1.57	1.051
		0.1563	1.19	0.79	1.57	47.667	0.149	54.801	1.03	0.79	1.19	47.600	0.091	64.405	1.92	1.195
		0.1875	0.46	0.46	0.46	45.833	0.000	54.293	0.46	0.46	0.46	45.833	0.000	66.847	0.46	1.333
	2.1	0.1250	0.00	**0.00**	0.00	64.167	0.000	60.835	0.00	**0.00**	0.00	64.167	0.000	69.524	1.07	1.103
		0.1563	0.00	0.00	0.00	53.833	0.000	52.455	0.00	0.00	0.00	53.833	0.000	61.553	0.00	1.116
		0.1875	0.00	0.00	0.00	56.667	0.000	54.155	0.00	0.00	0.00	56.667	0.000	67.839	0.00	1.304
0.75	1.5	0.1250	0.70	0.53	1.11	59.767	0.166	69.142	0.85	0.26	1.45	59.833	0.332	75.654	3.98	0.976
		0.1667	0.33	**0.00**	0.90	44.133	0.166	57.961	0.44	**0.00**	1.10	44.200	0.231	65.579	0.90	1.000
		0.2083	2.68	2.39	2.87	51.433	0.091	54.167	3.06	2.87	3.35	51.567	0.091	63.849	3.84	1.145
	1.8	0.1250	2.04	0.82	2.97	60.433	0.610	66.369	2.27	1.31	3.22	60.567	0.498	71.881	6.47	0.916
		0.1667	2.34	1.72	2.93	47.800	0.224	54.704	2.46	2.04	2.93	47.833	0.183	63.326	2.93	0.952
		0.2083	0.00	0.00	0.00	49.833	0.000	52.925	0.00	0.00	0.00	49.833	0.000	64.306	0.00	1.117
	2.1	0.1250	0.95	**0.00**	1.56	59.600	0.428	65.550	0.74	0.22	1.26	59.533	0.298	70.397	3.99	0.867
		0.1667	0.00	0.00	0.00	43.167	0.000	56.201	0.24	0.00	0.40	43.267	0.091	63.110	0.00	0.984
		0.2083	0.00	**0.00**	0.00	45.000	0.000	52.240	0.00	**0.00**	0.00	45.000	0.000	63.240	0.79	1.137

Table 4 continued

SF	NC	MRS	BRKGA 5 runs $p = 5 \times \lceil \frac{n \times n}{K} \rceil$, $p_e = 10\%$, $p_m = 30\%$, $\rho_e = 70\%$, $\mathcal{G} = \frac{n}{2}$												Almeida et al. (2016)	
			$\delta = PSS$						$\delta = Both$							
			Gap (%)			Makespan		Total time (CPU sec.)	Gap (%)			Makespan		Total time (CPU sec.)	Gap (%)	Total time (CPU sec.)
			Avg.	Min.	Max.	Avg.	Std.		Avg.	Min.	Max.	Avg.	Std.			
0.5	1.5	0.1250	1.79	0.26	3.41	61.633	0.833	81.355	1.96	0.80	2.87	61.733	0.584	85.962	6.17	0.749
		0.1625	0.20	**0.00**	1.01	49.933	0.224	67.694	0.34	**0.00**	0.68	50.000	0.183	73.764	4.27	0.796
		0.1875	0.51	0.51	0.51	41.667	0.000	64.173	0.10	**0.00**	0.51	41.533	0.075	72.743	0.88	0.890
	1.8	0.1250	0.79	0.34	1.19	60.267	0.240	72.780	0.97	0.63	1.53	60.367	0.257	77.886	5.19	0.658
		0.1625	0.29	**0.00**	1.07	55.467	0.240	70.753	0.41	**0.00**	0.98	55.533	0.224	75.775	2.42	0.872
		0.1875	0.14	**0.00**	0.36	47.900	0.091	62.440	0.22	**0.00**	0.36	47.933	0.091	70.642	0.36	0.846
	2.1	0.1250	0.42	**0.00**	1.41	71.967	0.447	78.988	0.39	0.24	0.61	71.967	0.183	81.704	5.00	0.713
		0.1625	0.00	**0.00**	0.00	55.167	0.000	65.061	0.00	**0.00**	0.00	55.167	0.000	73.334	0.93	0.809
		0.1875	0.00	0.00	0.00	56.000	0.000	57.938	0.00	0.00	0.00	56.000	0.000	66.694	0.00	0.783
var.	1.5	0.1250	3.12	1.58	4.21	52.467	0.611	60.477	3.45	2.19	5.59	52.633	0.743	68.199	7.55	0.765
		0.1667	6.75	6.56	7.04	47.233	0.091	53.469	6.90	6.56	7.41	47.300	0.183	61.253	7.41	0.939
		0.2083	0.49	0.49	0.49	37.667	0.000	49.560	0.49	0.49	0.49	37.667	0.000	60.592	1.42	1.049
	1.8	0.1250	0.91	**0.00**	1.48	54.333	0.348	63.006	1.47	0.93	2.10	54.633	0.332	69.311	5.16	0.872
		0.1667	2.08	2.08	2.08	45.667	0.000	55.702	2.21	1.67	2.85	45.733	0.240	62.642	4.02	1.002
		0.2083	0.71	0.51	1.01	43.733	0.091	49.464	0.99	0.00	1.49	43.833	0.240	59.584	0.00	1.093
	2.1	0.1250	0.38	0.32	0.64	63.700	0.075	66.694	0.32	0.32	0.32	63.667	0.000	73.867	2.46	0.916
		0.1667	0.52	**0.00**	0.73	48.900	0.166	54.834	0.67	0.41	0.73	48.967	0.075	63.657	0.33	0.911
		0.2083	0.00	0.00	0.00	54.667	0.000	47.308	0.00	0.00	0.00	54.667	0.000	60.410	0.00	1.033
Overall			1.10	0.79	1.44	52.250	0.159	60.008	1.19	0.86	1.56	52.292	0.169	68.105	2.65	0.986

values of MRS, which correspond to the hardest instances with a fewer resources, hence having larger population sizes and thus being more time consuming.

The last row of Table 4 presents the average results across all the 216 instances in *Set 1*. We observe that the overall average gap provided by the multi-pass heuristic was improved by the $\text{BRKGA}_{\text{PSS}}$ from 2.65 to 1.10% (average gaps) and to 0.79% (minimum gaps).

In terms of CPU time, the $\text{BRKGA}_{\text{PSS}}$ requires an average time of one minute, while the heuristic by Almeida et al. (2016) needs, on average, 1 s. However, the supremacy of the $\text{BRKGA}_{\text{PSS}}$ is clear in terms of the quality of the obtained gaps and 1 min of average time is still negligible when we look into the difficulty of the PSPFR.

Similar conclusions can be drawn from the results obtained when using $\text{BRKGA}_{\text{Both}}$. In fact, the $\text{BRKGA}_{\text{Both}}$ achieves better results than those obtained by the heuristic of Almeida et al. (2016) for 26 classes out of 36 classes of instances both in terms of average and minimum gaps. Regarding the maximum gaps, better results were obtained for 22 classes of instances out of 36 classes. The 9 classes of instances whose minimum gaps obtained by the $\text{BRKGA}_{\text{Both}}$ are highlighted in boldface indicate that the $\text{BRKGA}_{\text{Both}}$ was able to find the optimal value of all these 54 instances, but the heuristic of Almeida et al. (2016) was not.

In Table 4, we can also observe the supremacy of the $\text{BRKGA}_{\text{Both}}$ over the $\text{BRKGA}_{\text{PSS}}$ regarding average gaps for 4 classes of instances, regarding minimum gaps for also 4 classes of instances and in terms of maximum gaps for 7 classes of instances. Furthermore, the $\text{BRKGA}_{\text{Both}}$ was able to reach the optimal solutions for the 12 instances in the classes defined by $\text{SF} = \text{var.}, \text{NC} = 1.8, \text{MRS} = 0.2083$, and $\text{SF} = 0.5, \text{NC} = 1.5, \text{MRS} = 0.1875$, whereas this was not the case with the $\text{BRKGA}_{\text{PSS}}$. This behavior may be associated with the use of the SSS decoder which was also the responsible for the $\text{BRKGA}_{\text{Both}}$ requiring a slightly higher computational time than the $\text{BRKGA}_{\text{PSS}}$.

From Table 5 and considering the $\text{BRKGA}_{\text{PSS}}$, one can conclude that as the SF decreases, the gaps also decrease, while the computational time increases. This increase may be justified by the fact that instances with a smaller SF have fewer resources and thus originate larger population sizes. For instances having $\text{SF} = 0.5$, the $\text{BRKGA}_{\text{PSS}}$ produced solutions that correspond to an improvement of 6 and 22.8 times the results of Almeida et al. (2016), for average and minimum percentage gaps, respectively. Concerning NC, we observe that an increase in this parameter leads, as expected, to a reduction in the gaps of the BRKGA. The $\text{BRKGA}_{\text{PSS}}$ was able to reduce the minimum gaps of the instances having $\text{NC} = 2.1$ to nearly 0.0%.

The instances having $\text{MRS} = 0.1625$ were the ones where the $\text{BRKGA}_{\text{PSS}}$ produced the smallest gaps with values of 0.16% and 0% for average and minimum gaps, respectively. We point out that all these instances have $\text{SF} = 0.5$ and thus correspond to cases that were among the hardest to tackle by the procedure of Almeida et al. (2016). In this subset of "harder" instances, we also find those with small values of MRS, such as $\text{MRS} = 0.1250$, where the BRKGA originates a minimum gap of 0.83%. This correspond to a major improvement, since the heuristic of Almeida et al. (2016) produced gaps roughly six times higher.

Table 5 *Set 1*: Comparative analysis—Summary

		BRKGA 5 Runs $p = 5 \times \lceil \frac{n \times n}{K} \rceil$, $p_e = 10\%$, $p_m = 30\%$, $\rho_e = 70\%$, $\mathcal{G} = \frac{n}{2}$												Almeida et al. (2016)	
		$\delta = PSS$						$\delta = Both$							
		Gap (%)			Makespan		Total time (CPU sec.)	Gap (%)			Makespan		Total time (CPU sec.)	Gap (%)	Total time (CPU sec.)
		Avg.	Min.	Max.	Avg.	Std.		Avg.	Min.	Max.	Avg.	Std.			
SF	1	1.29	1.14	1.45	52.385	0.065	56.592	1.30	1.11	1.55	52.393	0.104	65.824	2.10	1.191
	0.75	1.00	0.61	1.37	51.241	0.187	58.807	1.12	0.74	1.52	51.293	0.191	66.816	2.55	1.010
	0.5	0.46	0.12	0.99	55.556	0.231	69.020	0.49	0.19	0.84	55.581	0.177	75.389	2.80	0.791
	var.	1.66	1.28	1.96	49.819	0.154	55.613	1.83	1.40	2.33	49.900	0.201	64.391	3.15	0.953
NC	1.5	2.15	1.78	2.60	49.683	0.194	60.968	2.25	1.82	2.87	49.736	0.245	68.738	4.19	0.994
	1.8	0.97	0.56	1.37	50.997	0.190	59.700	1.11	0.65	1.53	51.064	0.206	67.633	2.54	0.992
	2.1	0.19	0.03	0.36	56.069	0.093	59.355	0.20	0.10	0.28	56.075	0.054	67.944	1.21	0.973
MRS	0.1250	1.51	0.83	2.17	59.483	0.350	67.116	1.64	1.06	2.33	59.556	0.333	72.978	4.84	0.890
	0.1563	1.36	1.23	1.49	50.222	0.050	54.333	1.33	1.23	1.49	50.211	0.055	63.111	2.10	1.152
	0.1625	0.16	0.00	0.70	53.522	0.155	67.836	0.25	0.00	0.55	53.567	0.135	74.291	2.54	0.826
	0.1667	2.00	1.73	2.28	46.150	0.108	55.479	2.15	1.78	2.57	46.217	0.167	63.261	2.60	0.965
	0.1875	0.19	0.16	0.22	49.456	0.015	58.308	0.13	0.08	0.22	49.439	0.028	68.716	0.28	1.088
	0.2083	0.65	0.57	0.73	47.056	0.030	50.944	0.76	0.56	0.89	47.094	0.055	61.997	1.01	1.096
Overall		1.10	0.79	1.44	52.250	0.159	60.008	1.19	0.86	1.56	52.292	0.169	68.105	2.65	0.986

The $\text{BRKGA}_{\text{Both}}$ improve the results obtained by the $\text{BRKGA}_{\text{PSS}}$ for MRS $=$ 0.1563 and MRS $= 0.1875$, regarding average gaps and for $SF = 1$, MRS $= 0.1875$, and MRS $= 0.2083$ in terms of minimum gaps.

4.3.2 Results for the instances in Set 2

The heuristic of Almeida et al. (2016) provides the optimal makespan for five instances in *Set 2* and an upper bound on that value for the remaining ones. This is all the information that can be used for evaluating the performance of the *BRKGA* which we are proposing.

A closer look into the results of Almeida et al. (2016) reveals the unsuitability of using an off-the-shelf solver to tackle the mixed-integer linear programming formulation proposed by Correia et al. (2012) for the instances in *Set 2*. In fact, it was observed that, after 10 h of CPU time, CPLEX was unable to find a solution of at least the same quality of the one produced by the heuristic of Almeida et al. (2016) for 156 instances (more than 70% of the instances in *Set 2*). For the remaining 60 instances, the solver required, on average, 4861 s of CPU time to find a solution with at least the same objective value. Therefore, the development of efficient heuristics for the PSPFR is of great relevance and the CPU time which they consume shall be assessed considering the effort required by an alternative approach, such as the use of a solver. We will observe that the $\text{BRKGA}_{\text{PSS}}$ and $\text{BRKGA}_{\text{Both}}$ improve the solutions provided by the heuristic of Almeida et al. (2016), on average. Hence, we expect the solver to require an even greater CPU time to reach a solution of at least the same quality of that obtained by either BRKGA configuration.

To evaluate the results provided by the $\text{BRKGA}_{\text{PSS}}$ and by the $\text{BRKGA}_{\text{Both}}$ and due to the lack of information regarding the optimal solutions of almost all instances, we introduce a new concept referred to as *Performance Ratio* (PR). This is a relative ratio that we compute both for the makespan values (PR_{m}) and for the gaps (PR_{g}) as follows: $\text{PR}_{\text{m}} = \frac{Z^{\mathcal{B}^*} - Z^H}{Z^H} \times 100\%$, and $\text{PR}_{\text{g}} = \frac{D^{\mathcal{B}^*} - D^H}{D^H} \times 100\%$. In the previous expressions, $Z^{\mathcal{B}^*}$ ($D^{\mathcal{B}^*}$) denotes the best upper bound (minimum gap) provided by the $\text{BRKGA}_{\text{PSS}}$ or by the $\text{BRKGA}_{\text{Both}}$ and Z^H (D^H) denotes the upper bound (gap) obtained by the heuristic of Almeida et al. (2016).

Suppose that for some instance, and considering the $\text{BRKGA}_{\text{PSS}}$, we obtain $D^{\mathcal{B}^*}$=8% and D^H =10%.[1] In this case, PR_{g} = -20%, which indicates that the $\text{BRKGA}_{\text{PSS}}$ provides a gap 20% lower than the heuristic proposed by Almeida et al. (2016).

The analysis of improvements involving the makespan (PR_{m}) is also of great interest in particular when no lower bound besides the critical path length is available. In fact, the latter is often a poor lower bound for this problem (Correia et al. 2012), because it only considers information related to the precedence network.

Table 6 details the results for each class of instances. Each row of this table aggregates six instances. The results are then summarized in Table 7.

[1] The percentage gaps are computed as explained in Sect. 4.2, with the lower bound for each instance being equal to the length of the corresponding critical path.

Table 6 *Set 2*: Comparative analysis—detail

SF	NC	MRS		BRKGA 5 Runs $p = 5 \times \lceil \frac{n \times n}{K} \rceil$, $p_e = 10\%$, $p_m = 30\%$, $\rho_e = 70\%$, $\mathcal{G} = \frac{n}{2}$									Almeida et al. (2016)
				$\delta = PSS$					$\delta = Both$				
				Performance Ratio (%)		Makespan		Total time (CPU sec.)	Performance Ratio (%)		Makespan		Total time (CPU sec.)	Total time (CPU sec.)
				PR_m	PR_g	Avg.	Std.		PR_m	PR_g	Avg.	Std.		
1	1.5	0.0625		−5.92	−11.79	97.900	0.783	1158.783	−5.74	−11.30	98.167	0.859	1236.057	4.718
		0.0781		−4.36	−12.77	79.300	0.847	1478.504	−4.13	−12.06	79.300	0.789	1620.044	6.726
		0.0938	(1)	−3.06	−18.70	70.480	0.687	1791.673	−4.16	−26.96	70.200	0.837	1890.474	9.007
	1.8	0.0625		−6.49	−14.61	100.533	1.027	1136.121	−5.91	−13.31	100.800	0.704	1220.869	4.598
		0.0781		−5.17	−27.61	82.400	0.827	1455.465	−5.17	−26.56	82.067	0.763	1582.846	6.634
		0.0938	(1)	−2.41	−33.12	65.160	0.597	1776.367	−2.70	−34.66	65.040	0.649	1888.868	9.009
	2.1	0.0625		−5.33	−14.10	100.000	0.767	1132.525	−5.48	−14.29	99.867	0.753	1211.125	4.611
		0.0781		−3.02	−19.39	85.900	0.562	1451.702	−3.03	−20.18	86.267	0.718	1566.615	6.632
		0.0938		−1.68	−26.39	67.733	0.456	1726.694	−1.93	−29.17	67.633	0.582	1846.476	9.176
0.75	1.5	0.0625		−6.20	−12.67	110.300	0.656	1119.035	−6.57	−13.23	109.700	0.743	1243.254	3.356
		0.0792		−4.75	−14.07	81.967	0.865	1102.655	−5.29	−16.08	81.933	0.965	1192.030	4.085
		0.0938		−3.64	−16.33	69.800	0.611	1135.340	−4.04	−18.13	69.767	0.711	1246.575	5.213
	1.8	0.0625		−8.54	−16.92	110.733	1.067	1142.319	−9.75	−19.18	109.800	1.258	1282.192	3.335
		0.0792		−4.83	−19.95	83.767	0.959	1092.978	−5.25	−22.06	83.700	1.046	1183.419	4.182
		0.0938		−3.52	−27.59	70.800	0.872	1115.094	−3.24	−22.78	70.667	0.659	1226.932	5.101
	2.1	0.0625		−7.73	−20.69	108.967	1.108	1076.178	−8.20	−21.78	108.700	1.332	1181.252	3.257
		0.0792		−3.32	−19.42	77.400	0.673	1045.309	−3.94	−24.91	77.400	0.942	1138.075	4.005
		0.0938	(2)	−1.44	−18.13	71.150	0.537	1109.151	−1.80	−20.63	71.000	0.649	1218.183	4.940

Table 6 continued

SF	NC	MRS		BRKGA 5 Runs $p = 5 \times \lceil \frac{n \times n}{K} \rceil$, $p_e = 10\%$, $p_m = 30\%$, $\rho_e = 70\%$, $\mathcal{G} = \frac{n}{2}$									Almeida et al. (2016)
				$\delta = PSS$					$\delta = Both$				
				Performance Ratio (%)		Makespan		Total time (CPU sec.)	Performance Ratio (%)		Makespan		Total time (CPU sec.)	Total time (CPU sec.)
				PR_m	PR_g	Avg.	Std.		PR_m	PR_g	Avg.	Std.		
0.5	1.5	0.0625		−13.86	−25.82	120.767	2.408	1139.316	−15.88	−29.44	117.400	2.060	1231.441	2.468
		0.0781		−9.22	−20.41	98.433	1.393	1127.840	−10.34	−23.92	97.067	1.285	1217.346	2.965
		0.0938		−3.62	−24.49	81.500	0.829	1100.186	−3.44	−23.73	81.600	0.810	1170.960	3.231
	1.8	0.0625		−12.77	−25.88	129.200	1.712	1087.627	−12.46	−25.24	129.267	1.748	1179.768	2.398
		0.0781		−8.58	−27.97	88.167	1.193	1050.740	−8.23	−26.56	88.600	1.236	1124.984	2.820
		0.0938		−6.85	−29.21	77.567	0.939	1134.217	−7.10	−30.26	77.600	0.896	1224.682	3.489
	2.1	0.0625		−10.60	−22.31	120.933	1.628	1109.013	−12.16	−25.58	119.467	2.129	1179.548	2.510
		0.0781		−6.58	−22.85	87.100	0.716	1043.272	−7.88	−27.31	86.900	1.406	1112.957	2.796
		0.0938		−2.30	−35.18	75.400	0.316	1027.204	−2.52	−36.57	75.300	0.300	1108.814	3.085
var.	1.5	0.0625		−7.68	−13.21	110.367	1.572	1116.957	−9.34	−15.94	108.167	1.479	1351.140	3.286
		0.0792		−6.23	−24.12	79.533	1.077	1015.154	−5.86	−22.76	79.533	1.009	1120.748	3.778
		0.0938		−2.99	−30.28	70.467	0.745	1061.472	−3.19	−33.63	70.467	0.781	1176.741	4.731
	1.8	0.0625		−9.57	−21.46	111.067	1.703	1033.412	−10.64	−23.63	109.600	1.628	1230.622	3.075
		0.0792		−3.40	−14.15	77.967	0.719	1017.991	−3.78	−17.06	77.867	0.820	1113.140	3.820
		0.0938		−4.24	−19.31	71.400	0.787	1068.803	−3.76	−17.69	71.567	0.715	1181.989	4.906
	2.1	0.0625		−6.37	−17.71	109.433	1.290	997.032	−6.95	−19.13	108.733	1.254	1157.324	3.080
		0.0792		−3.08	−20.06	88.467	0.541	1052.956	−3.11	−20.73	88.433	0.732	1164.139	3.973
		0.0938	(1)	−2.74	−21.49	73.720	0.888	1018.186	−1.93	−15.64	74.000	0.578	1144.390	4.658
Overall			(5)	−5.69	−21.10	89.493	0.962	1169.095	−6.05	−22.25	89.148	1.003	1277.896	4.435

Table 7 *Set 2*: Comparative analysis—summary

			BRKGA 5 Runs $p = 5 \times \lceil \frac{n \times n}{K} \rceil$, $p_e = 10$, $p_m = 30$, $\rho_e = 70$, $\mathcal{G} = \frac{n}{2}$										Almeida et al. (2016)
			$\delta = PSS$					$\delta = Both$					
			Performance Ratio (%)		Makespan		Total time (CPU sec.)	Performance Ratio (%)		Makespan		Total time (CPU sec.)	Total time (CPU sec.)
			PR_m	PR_g	Avg.	Std.		PR_m	PR_g	Avg.	Std.		
SF	1	(2)	−4,22	−19.60	83.862	0.731	1443.826	−4.28	−20.56	83.862	0.739	1550.017	6.790
	0.75	(2)	−5.02	−18.43	87.827	0.827	1104.039	−5.48	−19.83	87.577	0.933	1212.213	4.164
	0.5		−8.26	−26.01	97.674	1.237	1091.046	−8.89	−27.62	97.022	1.319	1172.278	2.862
	var.	(1)	−5.19	−20.17	88.317	1.038	1042.898	−5.46	−20.78	87.853	1.008	1182.962	3.923
NC	1.5	(1)	−6.00	−18.72	89.499	1.044	1187.180	−6.53	−20.51	88.868	1.030	1299.865	4.464
	1.8	(1)	−6.42	−23.01	89.400	1.040	1167.471	−6.55	−23.09	89.217	1.015	1278.211	4.447
	2.1	(3)	−4.63	−21.57	89.583	0.796	1152.157	−5.04	−23.17	89.365	0.962	1254.966	4.394
MRS	0.0625		−8.42	−18.10	110.850	1.310	1104.026	−9.09	−19.34	109.972	1.329	1225.383	3.391
	0.0781		−6.15	−21.83	86.883	0.923	1267.920	−6.46	−22.77	86.700	1.033	1370.799	4.762
	0.0792		−4.27	−18.63	81.517	0.806	1054.507	−4.54	−20.60	81.478	0.919	1151.925	3.974
	0.0938	(5)	−3.28	−25.25	72.230	0.692	1247.489	−3.38	−25.98	72.206	0.681	1352.096	5.545
Overall			−5.69	−21.10	89.493	0.962	1169.095	−6.05	−22.25	89.148	1.003	1277.896	4.435

Tables 6 and 7 are organized as follows: columns 1–3 contain the characteristics of each class of instances; columns 5–9 present the results obtained by the $\text{BRKGA}_{\text{PSS}}$. In particular, columns 5 and 6 show the results associated with the PR_{m} and PR_{g}, respectively; columns 7–9 present the average makespan, standard deviation, and total CPU time required by the 5 runs, respectively; columns 10–14 refer to the results provided by the $\text{BRKGA}_{\text{Both}}$ and follow the above structure of 5–9; column 15 depicts the total time required by the heuristic of Almeida et al. (2016).

We note that the $\text{BRKGA}_{\text{PSS}}$ found the optimal solutions for all the five instances for which Almeida et al. (2016) also found the optimal value. These instances are indicated in column 4 and they were not used to compute the values presented in Tables 6 and 7.

From Table 6, we observe the $\text{BRKGA}_{\text{PSS}}$ performed better than the heuristic of Almeida et al. (2016) in all the 36 classes of instances. The best results in terms of PR_{m} were generally attained for the instances with smaller values of MRS, with a particular emphasis for the instances having $SF = 0.5$ and $\text{MRS} = 0.0625$. In fact, the class of instances which reported the highest improvement regarding PR_{m} (-13.86%) is actually defined by $SF = 0.5$, $\text{NC} = 1.5$, and $\text{MRS} = 0.0625$. The PR_{g} values seem to follow a different direction. This result may be related to the fact that smaller makespans are associated with higher values of MRS (thus allowing a greater degree of parallelization) for fixed values of SF and NC. In fact, the class of instances with $SF = 0.5$, $\text{NC} = 2.1$ and $\text{MRS} = 0.0938$ is the one associated with the smallest improvement in the makespan and the one having the greatest improvement in the gap. $\text{BRKGA}_{\text{Both}}$ improved the results of the $\text{BRKGA}_{\text{PSS}}$ for 25 out of the 36 classes of instances and the best results have been found in the same class of instances where the $\text{BRKGA}_{\text{PSS}}$ performed the best. The average time required to solve all the instances was 9% higher than the one required by the $\text{BRKGA}_{\text{PSS}}$ which, nonetheless, is still negligible considering both the difficulty of the PSPFR and the disastrous performance of a solver as reported in Almeida et al. (2016) and described at the beginning of this section. Looking into Table 7, we conclude that the quality of the results achieved by the $\text{BRKGA}_{\text{PSS}}$ increases and the computational effort decreases as the values of SF become smaller. Instances having $SF = 0.5$ are associated with higher values of standard deviation of makespan—as observed for the instances in *Set 1*. These instances can, in fact, be looked at as "difficult" to tackle. Moreover it is for these instances that the best results regarding both PR_{m} and PR_{g} were achieved.

Concerning the network complexity (NC), we observe that an increase in its value leads to a slight decrease in the CPU time. Instances having higher values of NC tend to be easier, because a smaller number of activities are likely to be processed in parallel due to the increased number of precedence relations. Amongst the different NC values, the largest improvements in terms of both PR_{m} and PR_{g} were achieved for the instances having $\text{NC} = 1.8$.

Regarding MRS, the best results for PR_{m} and PR_{g} are associated with $\text{MRS} = 0.0625$ and $\text{MRS} = 0.0938$, respectively. The latter can be explained by the fact that the instances with higher MRS are also the ones associated with smaller values of the average makespan. As discussed before, these instances typically allow a greater degree of parallelization, and hence, their corresponding optimal values can be closer to the value of the associated critical path length.

We observe overall values of $PR_m = -5.69\%$ and $PR_g = -21.10\%$ for the $BRKGA_{PSS}$. This represents a significant improvement over the results obtained by Almeida et al. (2016). The $BRKGA_{Both}$ achieved the best results for this set of instances with an overall $PR_m = -6.05\%$ and a $PR_g = -22.25\%$ and may hence be more appropriate for dealing with instances of large dimensions. In fact, the use of the two decoders may have contributed to a better exploration of the search space of feasible solutions which is larger for the instances in *Set 2*.

5 Conclusions

In this paper, we proposed a BRKGA for the PSPFR along with a new constructive heuristic for this problem based on a serial scheduling generation scheme (SSS). The BRKGA considers two decoding algorithms: the PSS heuristic proposed by Almeida et al. (2016) and the SSS.

Computational experiments were performed on 432 instances of the problem, partitioned into two sets. The computational results indicate that the BRKGA with the single decoder PSS provides the best results for the smaller sized instances, whereas the BRKGA with the two decoders, the PSS and SSS, achieves the best results for the set of larger instances. Furthermore, we could observe that the proposed approximate method is extremely robust in the sense that it has the ability of finding solutions of a similar quality for a given instance, in distinct runs.

Some directions for future research include the development of fast algorithms for deriving good lower bounds for the PSPFR. This would be of great relevance, since it would allow a more accurate evaluation of heuristics specially when it comes to tackling large-scale instances.

The fact that quite effective heuristic schemes exist for the PSPFR encourages the study of extensions of the problem investigated in this work. Some possibilities include non-homogeneous resources, multiple-skill contribution of a resource to the execution of an activity and preemption.

Another interesting direction for further research concerns the application to the PSPFR of a genetic algorithm whose chromosomes are encoded according to an activity list representation instead of random keys. This is something that requires using another genetic algorithm, since in a BRKGA, as the name indicates, all alleles are random values (random keys) in the interval [0,1].

Another aspect that we are neglecting in this work concerns the cost of the resources. The problem that we have studied emerges, for instance, in the context of software development and construction projects where the salary of the (human) resources is fixed and independent from the effort they put on the project. However, if this is not the case (e.g. in some consulting projects), then like for many project scheduling problem, accounting for time and cost may be altogether relevant for the decision-making process. This is an interesting and challenging research direction to explore.

Acknowledgements This work was supported by National Funding from FCT—Fundação para a Ciência e a Tecnologia, under the projects Fundação para a Ciência e a Tecnologia, UID/MAT/04561/2013 (CMAFCIO/FCUL) and UID/MAT/00297/2013 (CMA/FCT/UNL). The authors wish to thank the three anonymous referees for the valuable comments and suggestions provided which helped improving the manuscript.

References

Alcaraz J, Maroto C (2001) A robust genetic algorithm for resource allocation in project scheduling. Ann Oper Res 102(1):83–109

Alcaraz J, Maroto C (2006) A hybrid genetic algorithm based on intelligent encoding for project scheduling. In: Józefowska J, Węglarz J (eds) Perspectives in modern project scheduling, pp 249–274

Almeida BF, Correia I, Saldanha-da-Gama F (2015) An instance generator for the multi-skill resource-constrained project scheduling problem. Technical report, Faculdade de Ciências da Universidade de Lisboa—Centro de Matemática, Aplicações Fundamentais e Investigação Operacional. Available at https://ciencias.ulisboa.pt/sites/default/files/fcul/unidinvestig/cmaf-cio/SGama.pdf

Almeida BF, Correia I, Saldanha-da-Gama F (2016) Priority-based heuristics for the multi-skill resource constrained project scheduling problem. Expert Syst Appl 57:91–103

Brucker P, Drexl A, Möhring R, Neumann K, Pesch E (1999) Resource-constrained project scheduling: notation, classification, models, and methods. Eur J Oper Res 112(1):3–41

Correia I, Lourenço LL, Saldanha-da-Gama F (2012) Project scheduling with flexible resources: formulation and inequalities. OR Spectrum 34:635–663

Correia I, Saldanha-da-Gama F (2014) The impact of fixed and variable costs in a multi-skill project scheduling problem: an empirical study. Comput Ind Eng 72:230–238

Correia I, Saldanha-da-Gama F (2015) A modeling framework for project staffing and scheduling problems. In: Schwindt C, Zimmermann J (eds) Handbook on project management and scheduling, vol 1. Springer, Berlin, pp 547–564

Demeulemeester EL, Herroelen W (2002) Project scheduling: a research handbook. Springer, Berlin

Gonçalves JF, Mendes JJ, Resende MG (2008) A genetic algorithm for the resource constrained multi-project scheduling problem. Eur J Oper Res 189(3):1171–1190

Gonçalves JF, Resende MG (2011) Biased random-key genetic algorithms for combinatorial optimization. J Heuristics 17:487–525

Gonçalves JF, Resende MG (2013) A biased random key genetic algorithm for 2D and 3D bin packing problems. Int J Prod Econ 145(2):500–510

Gonçalves JF, Resende MG (2015) A biased random-key genetic algorithm for the unequal area facility layout problem. Eur J Oper Res 246(1):86–107

Gonçalves JF, Resende MG, Mendes JJ (2011) A biased random-key genetic algorithm with forward–backward improvement for the resource constrained project scheduling problem. J Heuristics 17(5):467–486

Gutjahr WJ, Katzensteiner S, Reiter P, Stummer C, Denk M (2008) Competence-driven project portfolio selection, scheduling and staff assignment. CEJOR 16:281–306

Hartmann S (2002) A self-adapting genetic algorithm for project scheduling under resource constraints. Naval Res Log (NRL) 49(5):433–448

Hartmann S, Briskorn D (2010) A survey of variants and extensions of the resource-constrained project scheduling problem. Eur J Oper Res 207(1):1–14

Hartmann S, Kolisch R (2000) Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem. Eur J Oper Res 127:394–407

Heimerl C, Kolisch R (2010) Scheduling and staffing multiple projects with a multi-skilled workforce. OR Spectrum 32:343–368

Herroelen WS, Demeulemeester EL (1998) Resource-constrained project scheduling: a survey of recent developments. Comput Oper Res 25:279–302

Klein R (2000) Bidirectional planning: improving priority rule-based heuristics for scheduling resource-constrained projects. Eur J Oper Res 127:619–638

Kolisch R (1996a) Efficient priority rules for the resource-constrained project scheduling problem. J Oper Manag 14:179–192

Kolisch R (1996b) Serial and parallel resource-constrained project scheduling methods revisited: theory and computation. Eur J Oper Res 90(2):320–333

Kolisch R, Hartmann S (1999) Heuristic algorithms for solving the resource-constrained project scheduling problem: classification and computational analysis. In: Weglarz J (ed) Project scheduling: recent models, algorithms and applications. Springer, Boston, pp 147–178

Kolisch R, Hartmann S (2006) Experimental evaluation of heuristics for the resource-constrained project scheduling problem: an update. Eur J Oper Res 174:23–37

Li H, Womer K (2009) Scheduling projects with multi-skilled personnel by a hybrid MILP/CP benders decomposition algorithm. J Sched 12:281–298

Li K, Willis R (1992) An alternative scheduling technique for resource constrained project scheduling. Eur J Oper Res 56:370–379

Mendes JJ, Gonçalves JF, Resende MG (2009) A random key based genetic algorithm for the resource constrained project scheduling problem. Comput Oper Res 36(1):92–109

Mingozzi A, Maniezzo V, Ricciardelli S, Bianco L (1998) An exact algorithm for the resource-constrained project scheduling problem based on a new mathematical formulation. Manag Sci 44(5):714–729

Özdamar L (1999) A genetic algorithm approach to a general category project scheduling problem. IEEE Trans Syst Man Cybern 29:44–69

Ruiz E, Albareda-Sambola M, Fernández E, Resende MG (2015) A biased random-key genetic algorithm for the capacitated minimum spanning tree problem. Comput Oper Res 57(C):95–108

Walter M, Zimmermann J (2016) Minimizing average project team size given multi-skilled workers with heterogeneous skill levels. Comput Oper Res 70(C):163–179

Węglarz J, Józefowska J, Mika M, Waligóra G (2011) Project scheduling with finite or infinite number of activity processing modes—a survey. Eur J Oper Res 208:177–205