OR Spectrum (2022) 44:575–602
https://doi.org/10.1007/s00291-021-00654-9

ORIGINAL ARTICLE

A partition-based branch-and-bound algorithm for the project duration problem with partially renewable resources and general temporal constraints

Kai Watermeyer[1] · Jürgen Zimmermann[1]

Received: 15 July 2020 / Accepted: 8 September 2021 / Published online: 9 December 2021

Abstract
The concept of partially renewable resources provides a general modeling framework that can be used for a wide range of different real-life applications. In this paper, we consider a resource-constrained project duration problem with partially renewable resources, where the temporal constraints between the activities are given by minimum and maximum time lags. We present a new branch-and-bound algorithm for this problem, which is based on a stepwise decomposition of the possible resource consumptions by the activities of the project. It is shown that the new approach results in a polynomially bounded depth of the enumeration tree, which is obtained by kind of a binary search. In a comprehensive experimental performance analysis, we compare our exact solution procedure with all branch-and-bound algorithms and state-of-the-art heuristics from the literature on different benchmark sets. The results of the performance study reveal that our branch-and-bound algorithm clearly outperforms all exact solution procedures. Furthermore, it is shown that our new approach dominates the state-of-the-art heuristics on well known benchmark instances.

Keywords Project scheduling · Branch and bound · Resource-constrained project scheduling · Partially renewable resources · Minimum and maximum time lags

✉ Kai Watermeyer
kai.watermeyer@tu-clausthal.de

Jürgen Zimmermann
juergen.zimmermann@tu-clausthal.de

[1] Operations Research Group, Institute of Management and Economics, Clausthal University of Technology, Julius-Albert-Str. 2, 38678 Clausthal-Zellerfeld, Germany

1 Introduction

Over the last decades, partially renewable resources have proven to be highly suitable to model constraints for a wide range of different real-life applications. The concept of partially renewable resources had first been introduced in Drexl et al. (1993) for a course scheduling problem and was later used in Drexl and Salewski (1997) to model constraints for a school timetabling problem. Further applications have been considered in Bartsch et al. (2006) and Knust (2010) for sports scheduling, in Briskorn and Fliedner (2012) for container transshipment, in Okubo et al. (2015) for machine scheduling, and in Androutsopoulos et al. (2020) for airport slot scheduling. Different modeling capabilities of partially renewable resources that cover logical relations between activities and further practical applications are discussed in Schirmer and Drexl (2001).

The concept of partially renewable resources allows to restrict the availability of a resource on an arbitrary subset of time periods of the whole planning horizon. As a consequence, the availability of a partially renewable resource for a specific time period is not fixed in advance, which provides a great flexibility for the scheduling process.

Böttcher et al. (1999) and Schirmer (1999) were the first to integrate the concept of partially renewable resources in the framework of project scheduling. Both works propose solution procedures for the classical resource-constrained project scheduling problem (RCPSP) with partially renewable resources (RCPSP/π). In Böttcher et al. (1999), the only available branch-and-bound algorithm for the RCPSP/π is given, which extends the basic enumeration scheme of Talbot and Patterson (1978) by two feasibilty bounds. Moreover, Böttcher et al. (1999) investigated different priority rules for a serial schedule generation scheme that considers, in compliance with the enumeration approach of Talbot and Patterson (1978), all feasible start times in each scheduling step. Further variants of a serial schedule generation scheme are provided in Schirmer (1999), which are based on a deterministic or randomized procedure to select the next activity to be scheduled and its start time. Additionally, Schirmer (1999) presents a tabu search algorithm and different techniques that are based on shift procedures to improve feasible solutions. Based on the work by Schirmer, Alvarez-Valdes et al. (2008) developed a GRASP algorithm for the RCPSP/π and a preprocessing procedure, in which the GRASP algorithm is contained. Different variants of the GRASP algorithm are investigated, where the best performance could be obtained by the combination with a path relinking approach. A further approximation method is given in Alvarez-Valdes et al. (2006) by a scatter search procedure that applies the GRASP algorithm for the generation of feasible solutions. Computational experiments could show that some variants of the scatter search procedure outperform the GRASP algorithm in terms of the solution quality, but at the expense of a strong increase in the computational effort (see, e.g., Alvarez-Valdes et al. 2015).

Recently, Watermeyer and Zimmermann (2020a) introduced an extension of the RCPSP/π that additionally considers minimum and maximum time lags between the start times of the activities in order to cover even more aspects of real-life projects. For this problem, which is denoted by RCPSP/max-π, Watermeyer and Zimmermann (2020a, 2020b) have proposed two branch-and-bound algorithms that are based on different enumeration schemes. The exact solution procedure in Watermeyer and Zimmermann (2020a) can be classified as a relaxation-based branch-and-bound algorithm that reduces the possible resource consumptions by the activities of the project step by step, where the resource-relaxation of the RCPSP/max-π represents the starting point. An alternative approach is considered in Watermeyer and Zimmermann (2020b), which schedules the activities of the project successively in combination with an unscheduling step.

In this paper, we present a new relaxation-based branch-and-bound algorithm for the RCPSP/max-π that decomposes the possible resource consumptions by the activities of the project in each enumeration step, which results in a partition of the start time domains of the activities. The new enumeration approach avoids the disadvantageous of the solution procedures in Watermeyer and Zimmermann (2020a, 2020b) that are mainly given by redundancies in the course of the enumeration and a great number of descendant nodes in each decomposition step.

The remainder of this paper is organized as follows. Section 2 provides a formal description of the RCPSP/max-π, where Sect. 3 deals with an example of a real-life project. Section 4 describes the enumeration scheme of the branch-and-bound algorithm that is discussed in Sect. 5. In Sect. 6, a comprehensive experimental performance analysis is provided, followed by some conclusions in Sect. 7.

2 Problem description

The RCPSP/max-π considers a set of activities $V := \{0, 1, \dots, n+1\}$, where each activity $i \in V$ has to be assigned a start time $S_i \in \mathbb{Z}_{\geq 0}$ under the consideration of temporal and resource constraints with the objective to minimize the project duration. Set V consists of real activities $i \in V^r$ with non-interruptible processing times $p_i \in \mathbb{Z}_{>0}$ and fictitious activities or events $i \in V^e$ with $p_i = 0$. The start and the end of the project are given by the fictitious activities 0 and $n+1$, respectively, where V might contain further events that represent milestones of the project. All activity pairs (i, j) with start-to-start precedence relationships are contained in set $E \subset V \times V$, where $\delta_{ij} \in \mathbb{Z}$ specifies the time lag that has to be hold between them, i.e., $S_j \geq S_i + \delta_{ij}$. It should be noted that the temporal constraints between the activities are given by minimum ($\delta_{ij} \geq 0$) and maximum time lags ($\delta_{ij} < 0$). For each project, a start time 0 and a maximum project duration (or completion time) $\bar{d} \in \mathbb{Z}_{\geq 0}$ are assumed, i.e., $S_0 = 0$ and $S_{n+1} \leq \bar{d}$. In the remainder of this paper, we call a sequence of start times $S = (S_i)_{i \in V}$ a schedule and speak of a time-feasible schedule S if $S \in \mathcal{S}_T$, where

$$\mathcal{S}_T := \{S \in \mathbb{Z}^{n+2}_{\geq 0} \mid S_j - S_i \geq \delta_{ij} \text{ for all } (i,j) \in E \wedge S_0 = 0\}$$

denotes the time-feasible region. Thereby, $(n+1, 0) \in E$ and $\delta_{n+1,0} = -\bar{d}$ are assumed, which means that the maximum project duration $\bar{d}$ is satisfied by each time-feasible schedule.

Besides the temporal constraints, the availabilities of the project resources have to be taken into account as well. The RCPSP/max-π considers partially renewable resources, where each resource $k \in \mathcal{R}$ is assigned to a subset $\Pi_k \subseteq \{1, 2, \dots, \bar{d}\}$ of all time periods of the planning horizon with a capacity $R_k \in \mathbb{Z}_{\geq 0}$. Each resource $k \in \mathcal{R}$ is only consumed by an activity $i \in V$ in each time period of Π_k the activity is in execution. The amount of capacity units of a resource k that is consumed by an activity i per time period is given by the resource demand $r^d_{ik} \in \mathbb{Z}_{\geq 0}$. Together with the number of periods in Π_k an activity i is in execution if it starts at time S_i, the so-called resource usage $r^u_{ik}(S_i) := |]S_i, S_i + p_i] \cap \Pi_k|$, we can calculate the resource consumption $r^c_{ik}(S_i) := r^u_{ik}(S_i) \cdot r^d_{ik}$. Conclusively, we can state for any schedule S the total resource consumption $r^c_k(S) := \sum_{i \in V} r^c_{ik}(S_i)$ of a resource $k \in \mathcal{R}$ by all activities of the project, so that the resource constraints can be expressed by $r^c_k(S) \leq R_k$ for all $k \in \mathcal{R}$. In the following, we say that a schedule S is resource-feasible if it satisfies all resource constraints, i.e., $S \in \mathcal{S}_R$ with the resource-feasible region

$$\mathcal{S}_R := \{S \in \mathbb{Z}^{n+2}_{\geq 0} \mid r^c_k(S) \leq R_k \text{ for all } k \in \mathcal{R}\}.$$

The RCPSP/max-π can be stated by

$$\left.\begin{array}{ll} \text{Minimize} & S_{n+1} \\ \text{subject to} & S \in \mathcal{S}_T \cap \mathcal{S}_R \end{array}\right\} \text{(P)}$$

with $\mathcal{S} := \mathcal{S}_T \cap \mathcal{S}_R$ as the feasible region. In the remainder of this paper, we call some schedule S that solves problem (P) optimal and denote by $\mathcal{OS}$ the set of all optimal schedules.

3 Numerical example

In what follows, we illustrate how the concept of partially renewable resources can be used for real-life projects. As an example, we consider a small project of a software company which is concerned with the development of a software for a customer. The order covers the development and the implementation of the software and has to be done within two weeks. The software company deploys an account manager (A) and a programmer (P) for the fulfillment of the order. Furthermore, a certain amount of data (D) for a cloud service, which is used to share data with the customer, is allocated to the project. For the account manager and the programmer, different contractual working conditions have to be taken into account. The account manager works at most five days a week and gets a weekend off after he had worked the whole weekend before. The programmer instead works not more than eight days and at most two weekend days over two consecutive weeks. The amount of data that is allocated to the project is 25 GB. Since the software company has to pay less for volumes of data at weekends, the guideline for each project is to use at least 40% of the whole data volume at weekend days. For the software company, the allocation of activities and resources on a daily basis seems to be sufficient, so that each period of the planning horizon corresponds to one day of the week, where the first period is assumed to be a Saturday (see Fig. 2).

Table 1 Overview of the resource specifications

$k \in \mathcal{R}$			Π_k	R_k
1	19	–	$\{1\}$	1
2	20	–	$\{2\}$	1
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
14	32	–	$\{14\}$	1
15	–	–	$\{1, 2, \ldots, 7\}$	5
16	–	–	$\{8, 9, \ldots, 14\}$	5
17	–	–	$\{1, 2, 8\}$	2
18	–	–	$\{1, 2, 9\}$	2
–	33	–	$\{1, 2, 8, 9\}$	2
–	34	–	$\{1, 2, \ldots, 14\}$	8
–	–	35	$\{3, \ldots, 7, 10, \ldots, 14\}$	15

Table 1 gives an overview of the partially renewable resources $\mathcal{R} = \{1, \ldots, 35\}$ that are used to model the restrictions as stated above. For a better understanding, we separate $\mathcal{R}$ into the sets $\widetilde{\mathcal{R}}_{\mathrm{A}} := \{1, 2, \ldots, 18\}$, $\widetilde{\mathcal{R}}_{\mathrm{P}} := \{19, 20, \ldots, 34\}$, and $\widetilde{\mathcal{R}}_{\mathrm{D}} := \{35\}$ which, respectively, contains all the resources which ensure that the restrictions of the corresponding resource $\sigma \in \{\mathrm{A}, \mathrm{P}, \mathrm{D}\}$ are satisfied. In Table 1, for the set Π_k of some resource $k \in \mathcal{R}$, the definition and its corresponding capacity R_k are both given in the same line. For example, from the first line, we get $\Pi_1 = \Pi_{19} = \{1\}$ and $R_1 = R_{19} = 1$ with $k = 1 \in \widetilde{\mathcal{R}}_{\mathrm{A}}$ and $k = 19 \in \widetilde{\mathcal{R}}_{\mathrm{P}}$. The restrictions for the account manager (A), the programmer (P), and the amount of data (D) are ensured by resources $k = 15, \ldots, 18$, $k = 33, 34$, and $k = 35$, respectively. For example, resource $k = 15$ restricts the working days of the account manager to at most five days in the first week and resources $k = 17, 18$ ensure that the account manager gets a weekend off after he had worked the whole weekend before. All other resources ($k = 1, \ldots, 14, 19, \ldots, 32$) establish that each employee can only be assigned to at most one activity per day, which is equivalent to a renewable resource constraint with a capacity of one unit.

In Table 2, all activities $i \in V$ which have to be executed to finish the project are listed with their corresponding processing time p_i and their resource demand for each resource $\sigma \in \{\mathrm{A}, \mathrm{P}, \mathrm{D}\}$, where $\widetilde{r}^d_{i\sigma} = a$ is used for simplification to state a resource demand $r^d_{ik} = a$ by activity $i \in V$ for all resources $k \in \widetilde{\mathcal{R}}_\sigma$. For example, activity $i = 3$ has a processing time $p_3 = 3$ and a resource demand $r^d_{3k} = 1$ for all resources $k \in \widetilde{\mathcal{R}}_{\mathrm{P}}$ and $r^d_{3k} = 5$ for all resources $k \in \widetilde{\mathcal{R}}_{\mathrm{D}}$. Figure 1 shows the project network which comprises the time lags δ_{ij} between the start times of all activities. The project starts with the consultation of the account manager with the customer to find out the specific requirements of the software ($i = 1$) which is followed by the discussion of the implementation with the programmer ($i = 2$) either directly after the consultation ($\delta_{12} = 3$) or one day after ($\delta_{21} = -4$). In the next step, the programmer implements the software which requires the access to the databases of the customer ($i = 3$). While the automated test runs ($i = 4$) to check the functionalities of the software and to reveal errors can not be started before the completion of activity

Table 2 Project activities

$i \in V$	Description	p_i	$\widetilde{r}^d_{i\mathrm{A}}$	$\widetilde{r}^d_{i\mathrm{P}}$	$\widetilde{r}^d_{i\mathrm{D}}$
0	Start of the project	0	0	0	0
1	Consultation with the customer	3	1	0	0
2	Discussion of the implementation	1	1	1	0
3	Data access and implementation	3	0	1	5
4	Automated test runs	2	0	0	4
5	On-the-spot system integration	1	1	1	2
6	Training course for employees	1	1	0	0
7	Termination of the project	0	0	0	0

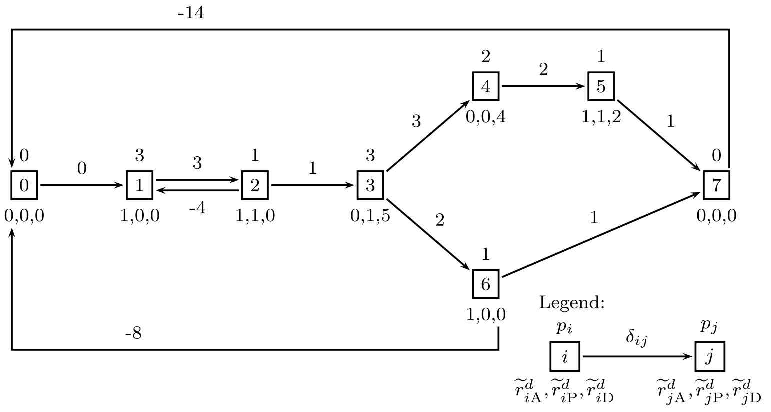
-14
0
0
0,0,0
0
3
1
1,0,0
3
-4
1
2
1,1,0
1
3
3
0,1,5
3
2
4
0,0,4
2
1
5
1,1,2
1
2
1
6
1,0,0
1
0
7
0,0,0
-8
Legend:
p_i
i
δ_{ij}
p_j
j
$\widetilde{r}^d_{i\mathrm{A}}, \widetilde{r}^d_{i\mathrm{P}}, \widetilde{r}^d_{i\mathrm{D}}$
$\widetilde{r}^d_{j\mathrm{A}}, \widetilde{r}^d_{j\mathrm{P}}, \widetilde{r}^d_{j\mathrm{D}}$

Fig. 1 Project network

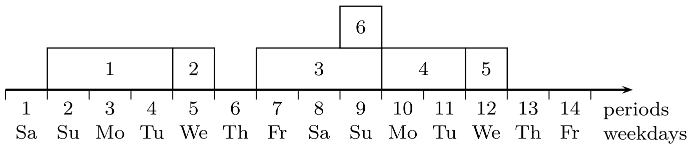

Fig. 2 Optimal schedule

$i = 3$ ($\delta_{34} = 3$), the training course for the employees ($i = 6$) of the customer can take place just two days after the start of activity $i = 3$ ($\delta_{36} = 2$). In order to have the possibility to offer more training for some employees who have difficulties with the software, the training course has to be completed before the end of the second weekend ($\delta_{60} = -8$). The last remaining step to complete the project is given by the system integration of the software and the correction of the latest detected errors directly at the customer's place of business ($i = 5$) which can be started directly after activity $i = 4$ ($\delta_{45} = 2$).

Figure 2 shows an optimal schedule $S^* = (0, 1, 4, 6, 9, 11, 8, 12)$ for the software project with a (minimal) project duration $S_7^* = 12$. It should be noted that activity $i = 3$ has to be executed over the whole second weekend due to the guideline that 40% of the whole required amount of data has to be used at the weekend. As a consequence, activity $i = 6$ has to be processed at Sunday of the second weekend, which prevents that activity $i = 1$ can be executed over the whole first weekend since this would imply that the account manager gets the second weekend off (see resources $k = 17, 18$).

4 Enumeration scheme

In this section, we present the enumeration scheme of our branch-and-bound algorithm. So far, only two different enumeration schemes are known for the RCPSP/max-π, which have been proposed by Watermeyer and Zimmermann (2020a, 2020b). These procedures can be categorized in a relaxation-based and a constructive approach. The relaxation-based branch-and-bound algorithm in Watermeyer and Zimmermann (2020a) reduces the possible resource usages of the activities of the project in each decomposition step. This procedure is continued until the optimal solution of the restricted resource-relaxation of problem (P) is either feasible or does not exist. An alternative approach has been developed in Watermeyer and Zimmermann (2020b), which is given by a constructive branch-and-bound algorithm that schedules the activities of the project successively in combination with an unscheduling step. The constructive branch-and-bound algorithm, which is inspired by the serial schedule generation scheme in Franck et al. (2001), makes use of insights with respect to the course of the resource usage of an activity dependent on its start time. Actually, for the first time, it could be shown that it is sufficient to consider a subset of all feasible start times of an activity in each scheduling step of a serial schedule generation scheme, so that the completeness of the procedure remains.

In this paper, we present a new relaxation-based enumeration scheme for the RCPSP/max-π. This approach mainly differs from classical procedures in project scheduling, which enumerate based on relaxations (see, e.g., Watermeyer and Zimmermann 2020a; De Reyck and Herroelen 1998; Fest et al. 1999), in the sense that the optimal solution of the relaxation is not removed by the decomposition step. The main advantages of the new approach compared to the existing enumeration schemes for partially renewable resources, which are described in detail below, are as follows. The depth of the enumeration tree is polynomially bounded from above, each decomposition step results in a constant number of new relaxations, and no feasible solution is generated more than once.

The basis of the new enumeration approach is a stepwise decomposition of the possible resource consumptions by the activities of the project with the resource-relaxation of problem (P) as the starting point. The decomposition scheme results in a partition of the start time domains of the activities. Accordingly, the enumeration approach can be illustrated by a directed outtree, where each node is represented by a vector $W := (W_i)_{i \in V}$ that assigns a start time domain $W_i \subseteq \{0, 1, \dots, \bar{d}\}$ to each activity of the project. In line with Watermeyer and Zimmermann (2020a, 2020b) we call W a start time restriction and say that W_i is the start time restriction of activity $i \in V$. Furthermore, we call $I := \{a, a+1, \dots, b\} \subseteq \mathbb{Z}$ a start time break of W_i if $I \subseteq \mathbb{Z} \setminus W_i$ and $a-1, b+1 \in W_i$, where $\mathcal{B}_i$ ($\mathcal{B}$) denotes the number of start time breaks of W_i (over all start time restrictions). The problem that has to be solved at each enumeration node corresponds to the resource-relaxation of problem (P) with additional unary constraints on the start times of the activities ($S_i \in W_i$). Accordingly, the problem at each enumeration node is given by

$$\left.\begin{array}{ll} \text{Minimize} & S_{n+1} \\ \text{subject to} & S \in \mathcal{S}_T(W) \end{array}\right\} \text{(R(W))}$$

with the feasible region $\mathcal{S}_T(W) := \{S \in \mathcal{S}_T \mid S_i \in W_i \text{ for all } i \in V\}$. In the following, we say that any schedule $S \in \mathcal{S}_T(W)$ is W-feasible and speak of a W-feasible start time S_i of an activity $i \in V$ if $\exists S' \in \mathcal{S}_T(W) : S'_i = S_i$. In Watermeyer and Zimmermann (2020a), it could be shown that $\mathcal{S}_T(W)$ has a unique minimal point, where a schedule $S \in \mathcal{S}_T(W)$ is called a minimal point of $\mathcal{S}_T(W)$ exactly if there is no other schedule S' with $S' \leq S$, i.e., $S'_i \leq S_i$ for all $i \in V$. Furthermore, Watermeyer and Zimmermann (2020a) have proposed an algorithm that either proves the unsolvability of (R(W)) or determines the unique minimal point of $\mathcal{S}_T(W)$ with a time complexity of $\mathcal{O}(|V||E|(\mathcal{B}+1))$. In the remainder of this paper, we assume that Algorithm 2 in Watermeyer and Zimmermann (2020a) is used to solve problem (R(W)), where we refer the reader to this reference for further details.

Next, we take a closer look at the enumeration scheme, which is outlined in Algorithm 1, where we discuss in particular how the start time domains of the activities are partitioned. For the sake of simplicity, we identify each enumeration node with its respective start time restriction W. In the first step of Algorithm 1, the root node W is initialized (if $\mathcal{S}_T \neq \emptyset$) by $W_i := \{ES_i, \dots, LS_i\}$ for all activities $i \in V$ with ES_i and LS_i as the earliest and latest time-feasible start times of activity i. Both the earliest and the latest time-feasible schedules ES and LS are determined by a classical label-correcting algorithm with a time complexity of $\mathcal{O}(|V||E|)$ (see, e.g., Ahuja et al. 1993, Sect. 5.4). Note that due to $\mathcal{S}_T(W) = \mathcal{S}_T$, problem (R(W)) corresponds to the resource-relaxation of problem (P) at the root node. Starting with $\Omega := \{W\}$ and $\Phi := \emptyset$, where Ω contains all unexplored nodes and Φ all generated feasible schedules, in each iteration some node W is removed from set Ω. For node W, the minimal point $S := \min \mathcal{S}_T(W)$ is determined or rather problem (R(W)) is solved. If $S \in \mathcal{S}_R$, a new feasible solution has been found and is stored as a candidate schedule in set Φ. Otherwise, there exists at least one resource $k \in \mathcal{R}$ whose capacity R_k is exceeded by the total resource consumption $r^c_k(S)$ of all activities, i.e, $k \in \mathcal{R}^c(S) := \{k' \in \mathcal{R} \mid r^c_{k'}(S) > R_{k'}\}$ with $\mathcal{R}^c(S)$ as the set of all conflict resources. In this case, it is checked if the existence of any feasible schedule in $\mathcal{S}_T(W)$ can be excluded, i.e., $\mathcal{S}(W) := \mathcal{S}_T(W) \cap \mathcal{S}_R = \emptyset$. For this, the minimum resource consumption $\underline{r}^c_k(W)$ for each conflict resources $k \in \mathcal{R}^c(S)$ is determined, where it is assumed that each activity $i \in V$ can be started at each time in W_i between its earliest and latest W-feasible start time. Accordingly, the minimum resource consumption for each activity $i \in V$ is given by $\underline{r}_{ik}^c(W) := \min\{r_{ik}^c(\tau) \mid \tau \in W_i \cap [ES_i(W), LS_i(W)]\}$ with $ES(W)$ and $LS(W)$ as the unique minimal and maximal point of $\mathcal{S}_T(W)$. It should be noted that $LS(W)$ is calculated with a time complexity $\mathcal{O}(|V||E|(\mathcal{B}+1))$ by an algorithm that has been proposed in Watermeyer and Zimmermann (2020a). If $\underline{r}_k^c(W) > R_k$ for any resource $k \in \mathcal{R}$, $\mathcal{S}(W) = \emptyset$ follows directly, so that node W represents a leaf of the enumeration tree. However, if the existence of any feasible schedule in $\mathcal{S}_T(W)$ cannot be excluded, the feasible region $\mathcal{S}_T(W)$ is decomposed. In the first step, a pair $(k, i) \in \mathcal{R}^c(S) \times V_k$ with $r_{ik}^u(S_i) > \underline{r}_{ik}^u(W)$ is chosen with $\underline{r}_{ik}^u(W) := \min\{r_{ik}^u(\tau) \mid \tau \in W_i \cap [ES_i(W), LS_i(W)]\}$. Note that for each conflict resources $k \in \mathcal{R}^c(S)$, there is always one activity $i \in V_k$ that satisfies condition $r_{ik}^u(S_i) > \underline{r}_{ik}^u(W)$, since otherwise the condition $\underline{r}_k^c(W) \leq R_k$ would not have been satisfied. The decomposition of the feasible region $\mathcal{S}_T(W)$ generates two descendant nodes, where W_i is partitioned into two start time restrictions W_i' and W_i'', i.e., $W_i = W_i' \cup W_i''$, $W_i' \cap W_i'' = \emptyset$ and $W_i', W_i'' \neq \emptyset$. Thereby, the partition of W_i is based on adding a lower and an upper bound on the resource usage of resource k by activity i for start time restrictions W_i'

Algorithm 1: Enumeration scheme

```
Input: Instance of problem RCPSP/max-$\pi$
Output: Set $\Phi$ of candidate schedules

1  Determine $ES$ and $LS$ with a label-correcting algorithm
2  if $\mathcal{S}_T = \emptyset$ then terminate
3  $W_i := \{ES_i, \dots, LS_i\} \quad \forall i \in V$
4  $\Omega := \{W\} \qquad \Phi := \emptyset$
5  while $\Omega \neq \emptyset$ do
6      Remove $W$ from set $\Omega$
7      $S := \min \mathcal{S}_T(W)$
8      if $S \in \mathcal{S}_R$ then
9          $\Phi := \Phi \cup \{S\}$
10     else if $\nexists k \in \mathcal{R} : \underline{r}_k^c(W) > R_k$ then
11         Select some pair $(k, i) \in \mathcal{R}^c(S) \times V_k$ with $r_{ik}^u(S_i) > \underline{r}_{ik}^u(W)$
12         Calculate $\bar{u}_{ik} := \left\lfloor \frac{r_{ik}^u(S_i) + \underline{r}_{ik}^u(W)}{2} \right\rfloor$
13         Initialize $W' := W$ and set $W_i' := \{\tau \in W_i \mid r_{ik}^u(\tau) > \bar{u}_{ik}\}$
14         Set $\Omega := \Omega \cup \{W'\}$
15         Initialize $W'' := W$ and set $W_i'' := \{\tau \in W_i \mid r_{ik}^u(\tau) \leq \bar{u}_{ik}\}$
16         if $\mathcal{S}_T(W'') \neq \emptyset$ then
17             Set $\Omega := \Omega \cup \{W''\}$
18 return $\Phi$
```

The total correctness of Algorithm 1 is established by Theorem 1 and Lemma 1, where it should be noted that Lemma 1 even shows that the depth of the enumeration tree is polynomially bounded from above by the input length of the considered instance.

Theorem 1 *Algorithm 1 is complete and sound, i.e., $\Phi \cap \mathcal{OS} \neq \emptyset$ if $\mathcal{OS} \neq \emptyset$ and $\Phi = \emptyset$ if $\mathcal{OS} = \emptyset$.*

Proof First, $\mathcal{S}_T(W) = \mathcal{S}_T \supseteq \mathcal{S}$ is given for the root node, so that the search space contains all feasible schedules. Next, it can be observed that Algorithm 1 ensures a decomposition of the search space of each node W with $S := \min \mathcal{S}_T(W) \notin \mathcal{S}_R$ which contains at least one feasible solution. Thereby, the decomposition step does not remove any (W-)feasible solution which can easily be derived from $\mathcal{S}_T(W) = \mathcal{S}_T(W') \cup \mathcal{S}_T(W'')$. Noticing that the number of decomposition steps is finite (Lemma 1) and that $S := \min \mathcal{S}_T(W)$ minimizes the project duration on set $\mathcal{S}_T(W)$, with $\mathcal{OS} \neq \emptyset \Rightarrow \mathcal{S} \neq \emptyset$ we can state $\mathcal{OS} \neq \emptyset \Rightarrow \boldsymbol{\Phi} \cap \mathcal{OS} \neq \emptyset$.

Finally, $\mathcal{OS} = \emptyset \Rightarrow \boldsymbol{\Phi} = \emptyset$ can directly be followed from the implication $\mathcal{OS} = \emptyset \Rightarrow \mathcal{S} = \emptyset$, which is given by the finiteness of $\mathcal{S}$, and the fact that each candidate schedule is feasible, i.e., $\boldsymbol{\Phi} \subseteq \mathcal{S}$. □

Lemma 1 *The maximum depth of the enumeration tree corresponding to Algorithm* 1 *is* $\mathcal{O}(|V||\mathcal{R}| \min(\overline{p}, |V||\mathcal{R}|\lceil \log_2 \overline{p} \rceil^2))$ *with* $\overline{p} := \max_{i \in V} p_i + 1$.

Proof As it can easily be verified, the generation of some node in Algorithm 1 is either associated with an increase in the minimum possible resource usage (W') or with a decrease in the maximum possible resource usage (W'') of some conflict resources $k \in \mathcal{R}^c(S)$ by an activity $i \in V_k$. Consequently, we can state a maximum number of $\mathcal{O}(|V||\mathcal{R}|\overline{p})$ decompositions which can be done until a leaf of the enumeration tree is reached.

Next, let $\mathcal{W} = (W^0, W^1, \ldots, W^\nu)$ be any sequence of nodes representing a path in the enumeration tree. First, we assume that all nodes $W^1, W^2, \ldots, W^\nu$ have been generated by increasing the minimum possible resource usage (W'). Then, $S = \min \mathcal{S}_T(W^0) = \min \mathcal{S}_T(W^1) = \ldots = \min \mathcal{S}_T(W^\nu) \notin \mathcal{S}_R$ can directly be followed. Hence, by taking into account that $\bar{u}_{ik}$ equals the (rounded) mean value of $r^u_{ik}(S_i)$ and the minimum possible resource usage in each decomposition step, we get that the maximum length ν of the path $\mathcal{W}$ is given by $\mathcal{O}(|V||\mathcal{R}|\lceil \log_2 \overline{p} \rceil)$. Since, as it can easily be verified, $\bar{u}_{ik}$ always equals at most the mean value of the minimum and the maximum possible resource usage of some resource $k \in \mathcal{R}^c(S)$ by an activity $i \in V_k$, any path $\mathcal{W}$ can contain at most $\mathcal{O}(|V||\mathcal{R}|\lceil \log_2 \overline{p} \rceil)$ nodes which are generated by a decrease in the maximum possible resource usage (W''). Accordingly, the maximum depth of the enumeration tree is $\mathcal{O}(|V|^2|\mathcal{R}|^2\lceil \log_2 \overline{p} \rceil^2)$. □

It should be noted that the partition-based decomposition, which we have described for Algorithm 1, only represents the best way for the partition we are aware of. In general, any partition would result in a total correct enumeration.

5 Branch-and-bound algorithm

The enumeration scheme in the previous section represents a general framework for the construction of a search tree for any instance of the RCPSP/max-π. In order to extend this scheme to a branch-and-bound procedure, we have to specify how the search tree is build. For this, we have to establish a strategy to determine which unexplored node in the search tree should be considered next (traversal strategy) and which pair (k, i) has to be chosen for the decomposition step (decomposition strategy).

The traversal strategy, which we use for our branch-and-bound algorithm, is an extension of the classical depth-first search strategy (DFS), called scattered-path search (SPS), which has been introduced in Watermeyer and Zimmermann (2020a). The SPS traverses the search tree exactly like the DFS, except that after a predefined time span an unexplored node with lowest level in the search tree and lowest lower bound on the project duration is chosen to be considered next. The idea of the SPS is to keep the advantage of the DFS to minimize the computing time to find a first feasible solution and simultaneously to avoid the drawback to get stuck in an unpromising part of the search tree.

For the decomposition of the feasible region of some node W with $S = \min \mathcal{S}_T(W) \notin \mathcal{S}_R$, the decomposition strategy establishes a priority rule to select some pair $(k, i) \in \mathcal{R}^c(S) \times V_k$. In what follows, we present the most promising priority rules we have investigated in preliminary computational tests. For the first priority rules, the conflict resource k and activity $i \in V_k$ are selected successively. In the first step, each conflict resource $k' \in \mathcal{R}^c(S)$ is assigned a priority value $\pi_{k'}$, with $\pi_{k'} := r^c_{k'}(S) - R_{k'}$ for the so-called absolute-capacity-overrun rule (ACO) and with $\pi_{k'} = (r^c_{k'}(S) - R_{k'})/R_{k'}$ for the relative-capacity-overrun rule (RCO). For both rules, the conflict resource with the greatest priority value and the smallest index is chosen for the decomposition, i.e., $k = \min\{k' \in \mathcal{R}^c(S) \mid \pi_{k'} = \max_{l \in \mathcal{R}^c(S)} \pi_l\}$. After the selection of the conflict resource k, each activity $i \in V_k$ with $r^u_{ik}(S_i) > \underline{r}^u_{ik}(W)$ is assigned a priority value π_i, where for both the ACO and RCO rule a priority value $\pi_i = r^c_{ik}(S_i) - \underline{r}^c_{ik}(W)$ is used. Like for the conflict resource k, the activity with the greatest priority value and the lowest index is chosen. Computational tests could reveal that the ACO and RCO rule, which tend to reduce the capacity overruns as fast as possible, are both well suited to solve small instances. In contrast, for greater instances, we were able to find even better priority rules that are based on the idea to select some pair $(k, i) \in \mathcal{R}^c(S) \times V_k$ that tends to reduce the maximal remaining capacity $R_k - \underline{r}^c_k(W')$ of the direct descendant node W' as much as possible. It should be noted that these priority rules can be seen to increase the probability that a descendant node W' can be removed from further considerations ($\underline{r}^c_k(W') > R_k$), which at least suggests a lower width of the search tree. The best of these priorities rules, called remaining-capacity-reduction rule (RCR), assigns a priority value

$$\pi_{k'i'} := \frac{(\bar{u}_{i'k'} + 1 - \underline{r}^u_{i'k'}(W))r^d_{i'k'}}{\max(R_{k'} - \underline{r}^c_{k'}(W), 0.1)}$$

to each pair $(k', i') \in \mathcal{R}^c(S) \times V_{k'}$ with $r^u_{i'k'}(S_{i'}) > \underline{r}^u_{i'k'}(W)$. Thereby, the nominator equals the minimum increase in $\underline{r}^c_{k'}(W)$ by the decomposition step with respect to node W' ($\underline{r}^c_{k'}(W')$) and the denominator corresponds to the maximal remaining capacity $R_{k'} - \underline{r}^c_{k'}(W)$ of resource k' (or 0.1 if $R_{k'} - \underline{r}^c_{k'}(W) = 0$). For the RCR rule, the pair (k', i') with the greatest priority value is chosen for the decomposition step, where ties are broken on the basis of lowest indices of the resources and activities.

After we have discussed how the search tree is build, we investigate different techniques that are used to avoid that unpromising parts of the search tree are explored. Besides the usage of lower and upper bounds on the project duration to prune the search tree, we have additionally implemented consistency tests to improve the performance of the branch-and-bound algorithm. In what follows, we sketch out briefly the consistency tests that are applied in the course of our branch-and-bound procedure. For further details we refer the reader to Watermeyer and Zimmermann (2020a), where all the consistency tests, we are concerned with in this section, have been introduced. All the following consistency tests have in common that they are used to reveal for some possible start time $t \in W_i$ of an activity $i \in V$ that there cannot exist any feasible schedule with $S_i = t$ in the feasible region of the currently considered node. Consequently, each deduced constraint by a consistency test can directly be expressed by a reduction rule for a start time restriction of some activity. As it is common practice, the different consistency tests described below are conducted iteratively one by one at each enumeration node until either no possible start time can be removed anymore or $W_i = \emptyset$ is given for some activity $i \in V$, i.e., a fixed point is reached.

The consistency tests, which are applied on the start time restriction W of the root node in a preprocessing step, are called temporal and W-interval consistency test. The temporal consistency test removes all times from the start time restriction W that are not W-feasible. Accordingly, the temporal consistency test for any possible start time $t \in W_i$ of some activity $i \in V$ can be expressed by the following condition and its reduction rule.

$$\nexists S \in \mathcal{S}_T(W) : S_i = t \quad \Rightarrow \quad W_i := W_i \backslash \{t\}$$

In order to remove possibly more start times from W, the W-interval consistency test takes also the resource constraints into account. For this, in a first step, the (indirect) minimum and maximum time lags $\widetilde{d}_{ij}(W, t)$ and $\widehat{d}_{ij}(W, t)$ between the W-feasible start times for all activity pairs $(i,j) \in V \times V$ are determined under the assumption that activity i starts at time t. Based on these time lags, a minimum resource consumption

$$r^{c,\min}_{ijkt}(W, \widetilde{D}, \widehat{D}) := \min\{r^c_{jk}(\tau) \mid \tau \in W_j \cap [t + \widetilde{d}_{ij}(W,t), t - \widehat{d}_{ij}(W,t)]\}$$

for any activity $j \in V_k \backslash \{i\}$ can be derived if activity i starts at time t. Consequently, each start time $t \in W_i$ of activity i can be removed if the sum of the minimum resource consumptions $r^{c,\min}_{ijkt}(W, \widetilde{D}, \widehat{D})$ over all activities $j \in V_k \backslash \{i\}$ together with $r^c_{ik}(t)$ exceeds the resource capacity R_k for some resource $k \in \mathcal{R}$. The condition and reduction rule for the W-interval consistency test for some activity $i \in V$ and any possible start time $t \in W_i$ can be stated by

$$\exists k \in \mathcal{R} : r^c_{ik}(t) + \sum_{j \in V_k \setminus \{i\}} r^{c,\min}_{ijkt}(W, \widetilde{D}, \widehat{D}) > R_k \quad \Rightarrow \quad W_i := W_i \setminus \{t\}.$$

Preliminary computational tests have shown that the calculation of the fixed point of the temporal and the W-interval consistency test at each node of the search tree would be too time-consuming. In addition, the tests could reveal that a restriction of the consistency tests to enumeration nodes with a low level in the search tree does not improve the performance. As a consequence, for all descendants of the root node, we apply consistency tests that can be performed in less computing time, but to the expense of a lower effectiveness (i.e., decrease in the number of start times which can be removed). The first of these consistency tests, which is called temporal-bound consistency test, eliminates for each activity $i \in V$ all start times from W_i that are lower than $ES_i(W)$ or greater than $LS_i(W)$.

$$t < ES_i(W) \vee t > LS_i(W) \quad \Rightarrow \quad W_i := W_i \setminus \{t\}$$

The second test, called resource-bound consistency test, can be seen as a simplified version of the W-interval consistency test that skips the temporal constraints between the activities. The condition and reduction rule of the resource-bound consistency test are given by

$$\exists k \in \mathcal{R}_i : r^c_{ik}(t) + \sum_{j \in V_k \setminus \{i\}} r^{c,\min}_{jk}(W) > R_k \quad \Rightarrow \quad W_i := W_i \setminus \{t\},$$

with $r^{c,\min}_{jk}(W) := \min\{r^c_{jk}(\tau) \mid \tau \in W_j\}$ and $\mathcal{R}_i := \{k \in \mathcal{R} \mid r^d_{ik} > 0\}$.

For the remainder of this section, it should be noted that all described consistency tests take the constraint $S_{n+1} < UB$ into account if they are applied in the course of the branch-and-bound algorithm, where UB equals the project duration of the best found solution or $\bar{d} + 1$ if no feasible solution has been found yet.

The branch-and-bound procedure is outlined in Algorithm 2. First, the preprocessing step is performed on the start time restriction W of the root node. Afterwards, given $\mathcal{S}_T(W) \neq \emptyset$, which means that $\mathcal{S}_T(W)$ could still contain some feasible solution, stack Ω is initialized and the upper bound UB on the shortest project duration is set to $UB := \bar{d} + 1$. In contrast to the enumeration scheme in Sect. 4, each node is represented by a triple (W, S, LB), which stores in addition the unique minimal point S of $\mathcal{S}_T(W)$ and the lower bound LB on the shortest project duration which is given by $LB := S_{n+1}$. In accordance with the SPS as described before, in each iteration of Algorithm 2 some node (W, S, LB) is removed from set Ω and it is checked by $LB < UB$ if its feasible region could contain a feasible solution with a lower project duration than UB. If this is the case, either the consistency tests from set Γ^T or Γ^B are applied on W, where Γ^T contains the temporal and the resource-bound consistency test, and Γ^B the temporal-bound and the resource-bound consistency test. In Sect. 6, it is shown that the right choice to use set Γ^T or Γ^B depends on whether instances with general temporal or classical precedence constraints are considered. Given $\mathcal{S}^{UB}_T(W) := \{S \in \mathcal{S}_T(W) \mid S_{n+1} < UB\} \neq \emptyset$ after the calculation of the fixed point, either of set Γ^T or Γ^B, schedule S is updated by $S := \min \mathcal{S}_T(W)$ if at least one possible start time of an activity has been removed. Since the minimal point of the feasible region S could be changed after the application of the consistency tests, the resource-feasibility of S is checked, where in case $S \in \mathcal{S}_R$ the new best solution is stored by $S^* := S$ and the upper bound on the shortest project duration UB is set to S^*_{n+1}. Noticing that the resource-bound consistency test is always assumed to be applied, the condition $\nexists k \in \mathcal{R} : \underline{r}^c_k(W) > R_k$ (see Algorithm 1) is assured for any node with $S \notin \mathcal{S}_R$. Therefore, the existence of at least one pair $(k, i) \in \mathcal{R}^c(S) \times V_k$ for any conflict resources k with $r^u_{ik}(S_i) > \underline{r}^u_{ik}(W)$, which is selected in accordance with the decomposition strategy, is guaranteed as well. The decomposition procedure is equivalent to Algorithm 1. While descendant node W' is always put on stack Ω, node W'' could represent a leaf of the search tree if $\mathcal{S}^{UB}_T(W'') = \emptyset$. If this is not the case, the minimal point S'' of $\mathcal{S}_T(W'')$ is calculated and its resource-feasibility is checked. Given $S'' \in \mathcal{S}_R$, S^* and UB are updated, while otherwise node (W'', S'', LB'') with $LB'' := S''_{n+1}$ is put on stack Ω to be explored in further iterations. After all nodes of the search tree have been explored, i.e., $\Omega = \emptyset$, the branch-and-bound algorithm either returns an optimal schedule S^* or shows that the considered instance is unsolvable ($\mathcal{S} = \emptyset$) with $UB = \bar{d} + 1$.

Algorithm 2: Branch-and-bound algorithm

```
Input: Instance of problem RCPSP/max-$\pi$
Output: Optimal schedule $S^*$

1  Determine $ES$ and $LS$ with a label-correcting algorithm
2  Set $W_i := \{ES_i, \ldots, LS_i\}$ for all $i \in V$
3  Apply the preprocessing step on $W$
4  if $\mathcal{S}_T(W) = \emptyset$ then terminate ($\mathcal{S} = \emptyset$)
5  Set schedule $S := \min \mathcal{S}_T(W)$ and lower bound $LB := S_{n+1}$
6  Initialize stack $\Omega := \{(W, S, LB)\}$ and upper bound $UB := \bar{d} + 1$
7  while $\Omega \neq \emptyset$ do
8      Remove triple $(W, S, LB)$ from stack $\Omega$ in accordance with SPS
9      if $LB < UB$ then
10         Apply consistency tests on $W$
11         if $\mathcal{S}^{UB}_T(W) \neq \emptyset$ then
12             Update $S := \min \mathcal{S}_T(W)$
13             if $S \in \mathcal{S}_R$ then
14                 $S^* := S$    $UB := S^*_{n+1}$
15             else
16                 Select some pair $(k, i) \in \mathcal{R}^c(S) \times V_k$ with $r^u_{ik}(S_i) > \underline{r}^u_{ik}(W)$ in
                     accordance with the decomposition strategy
17                 Calculate $\bar{u}_{ik} := \left\lfloor \frac{r^u_{ik}(S_i) + \underline{r}^u_{ik}(W)}{2} \right\rfloor$
18                 Initialize $W' := W$ and set $W'_i := \{\tau \in W_i \mid r^u_{ik}(\tau) > \bar{u}_{ik}\}$
19                 Set $S' := S$, $LB' := LB$, and put $(W', S', LB')$ on stack $\Omega$
20                 Initialize $W'' := W$ and set $W''_i := \{\tau \in W_i \mid r^u_{ik}(\tau) \leq \bar{u}_{ik}\}$
21                 if $\mathcal{S}^{UB}_T(W'') \neq \emptyset$ then
22                     Set $S'' := \min \mathcal{S}_T(W'')$
23                     if $S'' \in \mathcal{S}_R$ then
24                         $S^* := S''$    $UB := S^*_{n+1}$
25                     else
26                         Set $LB'' := S''_{n+1}$ and put $(W'', S'', LB'')$ on stack $\Omega$
27 if $UB = \bar{d} + 1$ then terminate ($\mathcal{S} = \emptyset$)
28 else return $S^*$
```

6 Performance analysis

In this section, we investigate the performance of our exact solution approach. For this, we present a comparison with all available branch-and-bound algorithms (BnB) and state-of-the-art heuristics from the literature that are concerned with partially renewable resources. The performance evaluation is based on different benchmark sets that cover instances with general temporal constraints (RCPSP/max-π) and classical precedence constraints (RCPSP/π). At the end of this section, in order to provide a starting point for the development of solution procedures, which are able to solve instances with renewable and partially renewable resources, we investigate to what extent our new BnB is able to solve instances with renewable resources.

The branch-and-bound algorithms, which are considered in this section, have all been coded in C++ and were compiled with the 64-bit Visual Studio 2017 C++-compiler. The computational experiments have been conducted on a single thread of an Intel Core i7-8700 CPU with 3.2 GHz and 64 GB RAM under Windows 10.

6.1 Comparison with branch-and-bound procedures

The settings of our BnB for the experimental performance analysis have been determined by preliminary computational tests and are given as follows. For the SPS, a time span of 5 s is used, consistency tests from set Γ^T (Γ^B) are applied for instances with general temporal (precedence) constraints, and the RCO (RCR) rule is chosen for instances with at most (more than) 15 real activities.

6.1.1 General temporal constraints

In a first step, we evaluate the performance of our BnB on benchmark set UBOπ, which has been generated by Watermeyer and Zimmermann (2020a) and provides to the best of our knowledge the only test instances for RCPSP/max-π. The instances of UBOπ were generated by a procedure that is described in Schirmer (1999, Sect. 10) based on instances of test set UBO, which has been generated by ProGen/max (see Schwindt 1998a; Kolisch et al. 1999). For details, we refer the reader to Watermeyer and Zimmermann (2020a). Test set UBOπ, which is accessible online[1], comprises 729 instances with $n = 10, 20, 50, 100, 200$ real activities, respectively, all of them with 30 partially renewable resources.

In what follows, we compare our exact solution procedure (BB1) with all available BnB for RCPSP/max-π that are, respectively, given by a constructive (BB2) and a relaxation-based approach (BB3) in Watermeyer and Zimmermann (2020b, 2020a). It should be noted that in Watermeyer and Zimmermann (2020a), BB3 has already been shown to outperform the mixed-integer linear programming solver IBM CPLEX based on a binary linear program. Table 3 shows the performance of all BnB that have been conducted with a time limit of 300 s on each test set UBO$n\pi$

[1] https://www.wiwi.tu-clausthal.de/abteilungen/betriebswirtschaftslehre-und-unternehmensforschung/forschung/benchmark-instances.

Table 3 Performance on benchmark set UBOπ (300 s)

		#nTriv	#opt	#feas	#uns	#unk	$\varnothing^{\text{cpu}}$ (s)	Δ^{lb} (%)	Δ^{gap} (%)
UBO10π	BB1	693	534	534	159	0	0.014	53.43	0.00
	BB2		534	534	159	0	0.065	53.43	0.00
	BB3		534	534	159	0	0.032	53.43	0.00
UBO20π	BB1	621	542	581	40	0	25.487	63.99	2.40
	BB2		537	581	40	0	28.086	64.67	2.27
	BB3		500	578	40	3	46.149	65.09	4.57
UBO50π	BB1	527	238	496	6	25	167.681	83.76	17.78
	BB2		183	491	5	31	198.016	88.16	20.00
	BB3		145	486	3	38	217.958	95.49	23.12
UBO100π	BB1	484	134	473	0	11	222.766	156.29	25.27
	BB2		85	472	0	12	249.827	168.68	29.11
	BB3		79	465	0	19	254.409	174.30	30.90
UBO200π	BB1	466	120	466	0	0	227.033	200.57	28.46
	BB2		93	446	0	20	244.808	226.70	33.57
	BB3		79	466	0	0	253.934	224.03	34.49

with $n = 10, 20, 50, 100, 200$ real activities. The results for BB2 and BB3 are taken from Watermeyer and Zimmermann (2020b, 2020a), noticing that all experiments have been conducted under the same conditions. The settings for BB2 and BB3 were both determined by preliminary computational tests dependent on the instance size. In fact, both BnB have been conducted with a different setting on each test set UBO$n\pi$ (see Watermeyer and Zimmermann 2020a, b).

The third column of Table 3 provides for each test set UBO$n\pi$ the number of instances for which the earliest schedule ES is not optimal (#nTriv), so-called nontrivial instances. In the remainder of this section, we restrict all our investigations to nontrivial instances only, regarding that each trivial instance can efficiently be solved to optimality. The following columns list the number of instances for which an optimal solution is found and verified (#opt), a feasible solution is detected (#feas), the unsolvability is shown (#uns), or the solvability status remains unknown (#unk). The last columns of Table 3 display the average computing time $\varnothing^{\text{cpu}}$, the average percentage deviation Δ^{lb} of upper bound UB from ES_{n+1}, and the average relative gap Δ^{gap} between UB and the best obtained lower bound LB on the shortest project duration by the BnB in relation to UB. For comparison purposes, the average values $\varnothing^{\text{cpu}}$, Δ^{lb}, and Δ^{gap} are given with respect to the number of all nontrivial instances, where the percentage deviation of UB from ES_{n+1} and the relative gap are defined by zero for all instances that have been proven to be unsolvable. Thereby, it should be noted that for all instances with unknown solvability status $UB = \bar{d} + 1$ is assumed (initialization step).

As it can be seen from Table 3, BB1 dominates both other exact solution procedures over the whole benchmark set UBOπ. While BB1 solves UBO10π faster than BB2 and BB3 and shows slightly better results for UBO20π, BB1 obtains superior results for all greater instances. However, it should be noted that BB2

Table 4 Results over all BnB (300 s)

instance set	$\#^{\cup}_{\text{opt}}$	$\#^{\cup}_{\text{feas}}$	$\#^{\cup}_{\text{uns}}$
UBO20π	549	581	40
UBO50π	241	496	6
UBO100π	135	473	0
UBO200π	121	466	0

Table 5 Impact of components on the performance for the RCR rule (UBO50π, 300 s)

	#opt	#feas	#uns	#unk	$\varnothing^{\text{cpu}}$ (s)	Δ^{lb} (%)	Δ^{gap} (%)
BnB (basic version)	212	495	5	27	182.530	89.03	29.25
+SPS	221	494	5	28	177.947	85.16	21.38
+Consistency tests	238	496	6	25	167.681	83.76	17.78

Table 6 Impact of components on the performance for the RCO rule (UBO50π, 300 s)

	#opt	#feas	#uns	#unk	$\varnothing^{\text{cpu}}$ (s)	Δ^{lb} (%)	Δ^{gap} (%)
BnB (basic version)	147	491	3	33	219.124	94.25	35.55
+SPS	160	494	3	30	211.817	88.12	26.37
+Consistency tests	202	494	5	28	187.724	85.06	19.16

determines a lower average relative gap Δ^{gap} than BB1 for test set UBO20π, where this advantage of BB2 is only given for UBO20π. Moreover, the results in Table 3 reveal a further advantage of BB1. While the settings of BB2 and BB3 are chosen with respect to the instance size, which has been shown to be decisive for the performance of both solution procedures, the settings of BB1 are only adapted for UBO10π to obtain lower computing times.

In order to investigate if the BnB have different strengths regarding the generation parameters of the test instances, Table 4 gives the number of instances for which at least one of the BnB could determine and verify an optimal solution ($\#^{\cup}_{\text{opt}}$), was able to find a feasible solution ($\#^{\cup}_{\text{feas}}$), or has proven its unsolvability ($\#^{\cup}_{\text{uns}}$).

Taking Table 3 into account, Table 4 reveals that BB1 determines the solvability status for each instance that could be solved to feasibility or has been shown to be unsolvable by at least one of the other BnB. Consequently, BB1 dominates both other solution procedures regarding the ability to determine the solvability status over all instances of benchmark set UBOπ. Furthermore, Table 4 shows that there is a small proportion of instances that were solved to optimality by BB2 or BB3 only. A closer look on the results reveals that for all these instances, except for one instance of test set UBO50π, only BB2 was able to find and verify an optimal solution. The advantage of the construction-based procedure BB2 is given for instances with a low cardinality of the period sets Π_k and a high number of resources that are demanded by each real activity on average. In contrast, an increasing advantage of BB1 in comparison with both other solution procedures over all test sets can especially be observed if the cardinality of the period sets Π_k is getting greater and the number of resources that are demanded by each real activity on average is decreasing.

Next, we investigate to what extend the strategy to construct the search tree and the applied consistency tests affect the performance of BB1. Table 5 shows the performance of BB1 on test set UBO50π with a time limit of 300 s, where the RCR rule is applied for the decomposition. The first line of Table 5 shows the performance of the basic version of BB1, i.e., BB1 is conducted with the traversal strategy DFS without any consistency test. The following lines provide the results if the stated component is additionally applied for BB1. To illustrate the impact of the selected decomposition strategy on the performance, Table 6 shows the results for BB1 if the RCO rule is applied instead. Tables 5 and 6 reveal that the performance of BB1 strongly depends on the selected decomposition strategy, where the improvement of the performance by the components is also strongly influenced by the decomposition strategy. In conclusion, while all components of BB1 are able to improve the performance, the decomposition strategy can be identified to be most decisive. It is also worth mentioning that already the basic version of BB1 (with the RCR rule) is able to determine the solvability status for more instances than both other BnB and can solve more instances as well.

6.1.2 Precedence constraints

In this section, we investigate the performance of BB1 on instances of the RCPSP/π or rather on instances with classical precedence constraints. In order to compare BB1 with all available BnB for RCPSP/π, we additionally consider the only BnB for RCPSP/π (BB4), which is discussed in Böttcher et al. (1999). The results for BB2, BB3, and BB4 are taken from Watermeyer and Zimmermann (2020b), where it should be noted that a reimplemented version of BB4 has been used for the computational experiments. BB2 and BB3 have been conducted with the settings for UBO20π or UBO50π, while BB4 was applied with both feasibility bounds from Böttcher et al. (1999).

Table 7 shows the performance of all BnB that are conducted with a time limit of 300 s on the Böttcher benchmark set, which has been provided by the authors of Böttcher et al. (1999) for the evaluation of heuristics in Alvarez-Valdes et al. (2006, 2008). The benchmark set comprises the test sets P10π, P15π, P20π, P25π, and P30π with 10, 15, 20, 25, and 30 real activities, respectively, all of them with 30 partially renewable resources. While P10π covers 2160 instances, all other test sets contain 250 instances, respectively. As it can be seen from Table 7, BB1 dominates all other BnB over all test sets of the Böttcher benchmark set, where BB1 solves or determines the solvability status for each test instance that is solved or whose solvability status is determined by at least one of the other BnB. Furthermore, the results of the analysis reveal that all the instances that could not be solved to optimality or for which the unsolvability could not be shown by BB1, BB2, or BB3 were all

Table 7 Performance on the Böttcher benchmark set (300 s)

		#nTriv	#opt	#feas	#uns	#unk	$\varnothing^{\text{cpu}}$ (s)	Δ^{lb} (%)	Δ^{gap} (%)
P10π	BB1	2108	827	827	1281	0	0.003	11.20	0.00
	BB2		827	827	1281	0	0.055	11.20	0.00
	BB3		827	827	1281	0	0.007	11.20	0.00
	BB4		827	827	1281	0	0.023	11.20	0.00
P15π	BB1	204	188	188	16	0	0.117	37.23	0.00
	BB2		188	188	16	0	1.704	37.23	0.00
	BB3		188	188	16	0	1.948	37.23	0.00
	BB4		181	181	16	7	12.717	38.19	1.47
P20π	BB1	165	144	145	17	3	10.143	50.60	1.05
	BB2		139	142	17	6	16.464	52.69	2.40
	BB3		139	142	17	6	16.920	52.66	2.40
	BB4		136	139	16	10	27.124	54.71	4.38
P25π	BB1	136	113	120	14	2	19.949	67.15	2.79
	BB2		112	116	14	6	22.155	69.31	3.56
	BB3		112	115	14	7	22.074	68.90	3.36
	BB4		105	111	11	14	46.535	74.32	7.90
P30π	BB1	122	104	108	8	6	24.632	90.05	4.35
	BB2		104	104	8	10	24.655	91.99	4.71
	BB3		104	104	8	10	24.615	91.99	4.74
	BB4		98	104	3	15	53.327	98.53	9.90

created with the same generation parameter setting that is denoted by "cell 25" in Böttcher et al. (1999, Table 8). In fact, the test sets P20π, P25π, and P30π contain only ten instances that are challenging for BB1, BB2, and BB3. These instances are specified by a great number of demanded resources by each real activity on average, a low scarcity of the resources, and a high cardinality of the sets Π_k, where not all periods in Π_k are consecutive.

The second benchmark set comprises the test sets J10π, J20π, J30π, J40π, and J60π, each containing 960 instances with 30 partially renewable resources and 10, 20, 30, 40, and 60 real activities, respectively. All instances of the benchmark set were generated by the procedure described in Schirmer (1999, Sect. 10) as a part of the instance generator ProGen/πx (Drexl et al. 2000). The test sets J10π, J20π, J30π, and J40π have been generated by Schirmer (1999) and were later complemented by test set J60π that has been generated in Alvarez-Valdes et al. (2006, 2008) for the evaluation of heuristics for the RCPSP/π. It should be noted that nine instances of J10π, whose unsolvability has been shown in Schirmer (1999, Sect. 10.4), are not part of the performance analysis since these instances could not be provided to us. Table 8 shows the performance on the so-called Schirmer-Alvarez-Valdes (SAV) benchmark set, where all BnB have been conducted with a time limit of 300 s. It can be observed that BB1 significantly dominates all other BnB over all instances of the benchmark set. In fact, BB1 can considerably increase the number of optimal solved instances for the test sets J30π, J40π, and J60π. Furthermore, BB1 solves all

Table 8 Performance on the SAV benchmark set (300 s)

		#nTriv	#opt	#feas	#uns	#unk	$\varnothing^{\text{cpu}}$ (s)	Δ^{lb} (%)	Δ^{gap} (%)
J10π	BB1	808	803	803	5	0	0.047	10.37	0.00
	BB2		803	803	5	0	0.063	10.37	0.00
	BB3		803	803	5	0	0.060	10.37	0.00
	BB4		802	802	5	1	0.541	10.54	0.07
J20π	BB1	565	565	565	0	0	0.121	5.19	0.00
	BB2		563	565	0	0	1.965	5.22	0.07
	BB3		564	565	0	0	2.313	5.22	0.05
	BB4		509	561	0	4	35.533	7.93	3.25
J30π	BB1	453	452	453	0	0	1.388	3.75	0.04
	BB2		431	453	0	0	17.603	4.37	0.86
	BB3		427	453	0	0	20.723	4.18	0.78
	BB4		345	435	0	18	75.767	14.18	7.50
J40π	BB1	386	378	386	0	0	8.354	4.33	0.13
	BB2		347	386	0	0	35.153	6.06	1.95
	BB3		341	386	0	0	38.599	6.07	1.97
	BB4		261	363	0	23	100.109	21.24	10.63
J60π	BB1	346	314	346	0	0	31.032	6.05	1.32
	BB2		269	346	0	0	73.353	8.91	3.91
	BB3		268	346	0	0	69.313	14.06	5.90
	BB4		186	309	0	37	140.073	40.50	17.37

instances of J20π, provides the lowest computing times for test set J10π, and solves all instances for which at least one of the other BnB could find and verify an optimal solution. A closer look on the generation parameters for the test instances reveals that the increasing advantage of BB1 compared to all other BnB can especially be observed if the cardinality of the period sets Π_k and the number of demanded resources by each real activity on average are getting greater and the number of intervals, on which the periods in Π_k are consecutive without any interruption, is decreasing.

In order to evaluate the performance of the BnB for greater instances with precedence constraints (RCPSP/π), we have extended the SAV benchmark set by creating instances with 100 and 200 real activities and 30 partially renewable resources, denoted by J100π and J200π in the following. For the generation of the project network and the activity durations, we have used the instance generator ProGen/max (cf. Schwindt 1998a) and applied the procedure in Schirmer (1999, Sect. 10) to create the partially renewable resources. At first, we created test instances with the same settings as for the SAV benchmark set, which result in test sets with only trivial instances. It should be noted that although a small number of nontrivial instances was to be expected, the strong increase in the number of trivial instances rather suggests that ProGen/max in combination with the reimplemented procedure of Schirmer (1999) tend to create less restrictive instances than the original procedure. As a consequence, we changed the values of the generation parameters, which

Table 9 Performance on test sets J100π and J200π (300 s)

		#nTriv	#opt	#feas	#uns	#unk	$\varnothing^{cpu}$ (s)	Δ^{lb} (%)	Δ^{gap} (%)
J100π	BB1	716	542	685	4	27	82.476	4.41	2.76
	BB2		357	666	4	46	157.428	9.18	6.58
	BB3		311	659	4	53	178.982	15.04	9.96
	BB4		41	57	4	655	281.624	59.96	33.48
J200π	BB1	639	510	623	0	16	69.686	2.66	2.00
	BB2		320	610	0	29	158.335	7.40	5.72
	BB3		307	601	0	38	168.371	11.52	8.04
	BB4		33	42	0	597	285.030	59.81	33.73

are given by the order strength (OS), the resource factor (RF), the resource strength (RS), the horizon factor (HF), the cardinality factor (CF) and the interval factor (IF). For a detailed description of the generation parameters, we refer the reader to Schirmer (1999, Sect. 10). In a first step, we assigned the values $\{0.3, 0.7\}$ to each parameter and generated for each combination of the parameter values ten instances with 100 real activities (640 instances). Since this test set contained more than 50% trivial or unsolvable instances, we analyzed the parameters with the greatest impact on the number of trivial and unsolvable instances that were given by CF and RS. In order to determine value combinations for CF and RS with a lower proportion of trivial and unsolvable instances, we generated a new test set with 100 real activities for which we chose the values $\{0.1, 0.3, 0.5, 0.7, 0.9\}$ for CF and RS, respectively, with the same settings for all other parameters as before (4000 instances). After that, we determined all parameter value combinations of CF and RS for which the proportion of trivial instances was lower than 50% and for which at least 60% of all instances (trivial instances included) could be solved to feasibility by BB1, BB2, or BB3 within 300 s. The only parameter value combination, which satisfied these conditions, was given by CF = 0.7 and RS = 0.7 that was finally chosen for the generation of the test sets J100π and J200π. For all other generation parameters, the values OS $\in \{0.3, 0.7\}$, HF $\in \{0.3, 0.5, 0.7, 0.9\}$, RF $\in \{0.1, 0.3, 0.5, 0.7\}$ and IF $\in \{0.1, 0.5, 0.9\}$ were selected. The test sets J100π and J200π are available online.[2]

Table 9 shows for all BnB the performance on the test sets J100π and J200π with a time limit of 300 s. For both BB2 and BB3, we chose the settings that have been used for the test set UBO100π (cf. Watermeyer and Zimmermann 2020a, b) since the settings for J60π result in more than 50% of the nontrivial instances of test set J200π for which the solvability status could not be determined. It should be noted that this observation demonstrates once more the advantage of BB1 that the setting of BB1 does not have to be adjusted dependent on the instance size, which has been shown to be decisive for BB2 and BB3 (see Watermeyer and Zimmermann 2020a, b).

[2] https://www.wiwi.tu-clausthal.de/abteilungen/betriebswirtschaftslehre-und-unternehmensforschung/forschung/benchmark-instances.

Table 10 Comparison of BB1 with heuristics on the SAV benchmark set

		#inst	Time limit				Heuristics	
			10 s	30 s	60 s	300 s	SS	GR
J10π	$\#_{\neq}^{\text{opt}}$	803	0	–	–	–	0	1
	Δ^{opt}		0.000%	–	–	–	0.00%	0.00%
	$\varnothing^{\text{cpu}}$		0.054 s	–	–	–	2.5 s	0.3 s
J20π	$\#_{\neq}^{\text{opt}}$	565	0	0	0	–	5	19
	Δ^{opt}		0.000%	0.000%	0.000%	–	0.04%	0.12%
	$\varnothing^{\text{cpu}}$		0.086 s	0.127 s	0.139 s	–	17.0 s	1.2 s
J30π	$\#_{\neq}^{\text{opt}}$	453	3	1	1	1	21*	33*
	Δ^{opt}		0.046%	0.010%	0.002%	0.002%	0.10%*	0.21%*
	$\varnothing^{\text{cpu}}$		0.327 s	0.600 s	0.903 s	1.586 s	28.8 s	3.7 s
J40π	$\#_{\neq}^{\text{opt}}$	386	14	10	6	4	31*	54*
	Δ^{opt}		0.109%	0.058%	0.030%	0.022%	0.25%*	0.59%*
	$\varnothing^{\text{cpu}}$		0.711 s	1.754 s	2.899 s	9.548 s	51.1 s	7.2 s
J60π	$\#_{\neq}^{\text{opt}}$	346	41	37	33	28	67	80
	Δ^{opt}		0.969%	0.553%	0.489%	0.288%	0.71%	1.16%
	$\varnothing^{\text{cpu}}$		1.748 s	4.676 s	8.573 s	35.465 s	175.9 s	13.4 s

From Table 9, it can be seen that the dominance of BB1 over all other BnB is getting even greater if the size of the test instances increases, where the main reason for this may be assumed to be given by the more restrictive resource constraints compared to the SAV benchmark set. As for all other test sets, BB1 determines for J100π and J200π the solvability status for each instance that could be solved to feasibility or has been shown to be unsolvable by at least one of the other BnB. Only for J100π, BB2 was able to solve one further instance to optimality. Interestingly, in contrast to the SAV benchmark set, the advantage of BB1 to all other BnB tend to increase if the number of demanded resources by each real activity on average is getting lower, while the observation remains, that the decrease in the number of intervals, on which the periods in Π_k are consecutive without any interruption, result in an increasing better performance of BB1.

6.2 Comparison with state-of-the-art heuristics

Based on the promising results, the question arises if BB1 can also compete with approximation methods. In the following, we compare our branch-and-bound algorithm BB1 with the state-of-the-art heuristics for the RCPSP/π that are given by a scatter search (SS) and a GRASP algorithm (GR) developed by Alvarez-Valdes et al. (2006, 2008). It should be noted that the following investigations are restricted to instances with precedence constraints, since for the RCPSP/max-π no approximation method is available.

Table 10 shows the performance of BB1, SS, and GR on the SAV benchmark set, where BB1 has been conducted with different time limits. The results of SS and GR are taken from Table 11.7 in Alvarez-Valdes et al. (2015). For a better overview, we only compare BB1 with the best performing variants of the two heuristics with respect to their solution qualities, where it should be noted that the other variants of SS and GR that are reported in Alvarez-Valdes et al. (2015) need lower computing times. At the end of this section, we will come back to this topic to take all variants into consideration.

Table 10 gives for each test set the number of instances for which the respective solution procedure could not determine the optimal solution ($\#^{\text{opt}}_{\neq}$), the average percentage deviation from the optimal solution (Δ^{opt}), and the average computing time ($\varnothing^{\text{cpu}}$) over all nontrivial and solvable instances (#inst). The symbol "–" indicates that BB1 has already proven the optimality for all instances of the test set with a lower time limit that is given in the table. To determine the optimal solutions for all instances, we have solved a time-indexed formulation for the RCPSP/π based on the binary linear program in Böttcher et al. (1999) with the mixed-integer linear programming solver IBM CPLEX (12.8.0) using multithreading. While we were able to solve all instances to optimality, the results for SS and GR in Table 10, marked with "$*$", are possibly not determined in relation to the optimal solutions, since Alvarez-Valdes et al. (2015) could not verify the optimality of the solutions for one instance of J30π and five instances of J40π. The solution procedures SS and GR were coded in C++ and conducted on a Pentium IV processor with 2.8 GHz. In order to ensure a fair comparison between the solution procedures, we multiplied the computing times of BB1 by $8/7$ corresponding to the clock pulse ration of the different workstations $(3200/2800 = 8/7)$.

The results in Table 10 show that BB1 determines with a time limit of 10 s more optimal solutions than SS and GR and obtains lower average percentage deviations Δ^{opt} for all test sets with less than 60 real activities, while BB1 outperforms both heuristics as well on test set J60π with a time limit of 30 s. It is especially worth mentioning that BB1 can achieve much better solution qualities than SS within a significant lower computing time. Moreover, it should be emphasized that all other variants of SS and GR, which are not part of Table 10, are clearly outperformed by BB1 as well. For BB2 and BB3, further investigations could show that they are only able to achieve better results than SS and GR for test sets J10π and J20π, while for all greater instances already GR outperforms the solution qualities of both BnB with significant lower computing times.

The comparison of BB1 with SS and GR for the Böttcher benchmark set has shown that the preprocessing procedure that is used for both heuristics is already able to determine more feasible solutions for the test sets P20π, P25π, and P30π. As a consequence, it would be interesting for future investigations to use the preprocessing procedure for BB1 as well, to be able to compare the solutions with SS and GR.

It should be noted that the results in this section indicate that serial schedule generation procedures, on which SS and GR are based on, seem not to be competitive with the relaxation-based enumeration scheme of our BnB. As a consequence,

Table 11 Performance of BB1 on instances with renewable resources (300 s)

	#inst	#opt	#feas	#uns	#unk	$\varnothing^{\text{cpu}}$ (s)	Δ^{lb} (%)	Δ^{gap} (%)
PAT	110	110	110	0	0	1.633	18.04	0.00
KSD30	480	465	480	0	0	14.651	13.41	1.02
KSD60	480	382	480	0	0	67.716	14.01	6.32
KSD90	480	357	480	0	0	84.235	20.64	9.11
KSD120	600	161	538	0	62	229.629	127.50	30.01
CV	623	37	623	0	0	291.387	146.32	43.93
UBO10	90	73	73	17	0	0.070	18.38	0.00
UBO20	90	70	70	19	1	6.565	23.19	0.74
UBO50	90	49	73	14	3	113.919	44.55	12.04
UBO100	90	25	48	6	36	223.410	154.44	37.80
UBO200	90	6	11	3	76	283.052	336.64	65.90
SM10	270	187	187	83	0	0.119	21.93	0.00
SM20	270	179	184	85	1	12.340	17.55	0.95
SM30	270	162	183	83	4	36.732	25.77	4.39

generation schemes for the project duration problem with partially renewable resources should focus in the future on our new partition-based enumeration approach.

6.3 Outlook

The concept of partially renewable resources is well known to generalize classical renewable resources (cf. Böttcher et al. 1999). However, the rather theoretical characteristic of this observation has not been emphasized in the literature so far. As a consequence, it might seem to be promising to apply solution procedures for partially renewable resources for instances with classical renewable resources. In fact, this approach can only be expected to be reasonable for small instances, since the procedure to transform the renewable resources to partially renewable resources (cf. Böttcher et al. 1999) leads to a pseudo-polynomial growth in the instance size.

Despite the stated restriction, nevertheless, it seems to be interesting to investigate the performance of BB1 on small instances with renewable resources. Furthermore, the question should be answered, up to which instance size BB1 is able to obtain a reasonable performance. For this, we have tested BB1 on test instances of the well known benchmark sets PAT (cf. Patterson 1984) and KSD (cf. Kolisch et al. 1995) for the RCPSP and test sets UBO (cf. Franck et al. 2001) and SM (cf. Kolisch et al. 1999) for the RCPSP/max. Additionally, we have considered the new test set CV, which has been generated by Coelho and Vanhoucke (2020) to provide intractable instances for the RCPSP. In fact, it has been shown that although each test instance of CV contains only up to 30 real activities, different

Table 12 Performance of BB1 on benchmark set CV

t^{lim} (s)	#opt	$\varnothing^{\text{cpu}}$ (s)	Δ^{lb} (%)	Δ^{gap} (%)	Δ^{ub} (%)	$\#^{\text{ub}}_{<}$	$\#^{\text{ub}}_{=}$
300	37	291.387	146.32	43.93	1.64	2	274
600	59	567.835	145.33	41.44	1.24	3	332
1200	98	1089.745	144.69	38.07	0.97	3	375
3600	174	2952.604	143.84	32.25	0.63	5	445

variants of a BnB and the solver IBM CPLEX based on a time-indexed formulation were not able to solve any of the test instances after 20 h runs.

For the experimental study, BB1 has been conducted with the delayed-start-time rule (DST) (cf. Watermeyer and Zimmermann 2020a) to choose the conflict resource $k \in \mathcal{R}^c(S)$ and the activity $i \in V_k$ in each decomposition step, which has been shown to be promising for renewable resources in preliminary computational tests. All other settings of BB1 were selected in accordance with the RCPSP/π-instances.

Table 11 shows the results of the performance analysis over all instances (#inst, trivial instances included) for a maximum run time of 300 s, while Table 12 provides further investigations on benchmark set CV for greater time limits (t^{lim}) due to its intractability. Thereby, it should be noted that only the benchmark sets PAT and KSDn with $n = 30, 60, 90$ real activities contain trivial instances, where PAT covers three of them and the KSD benchmark sets 120, respectively. Table 12 gives in addition the mean percentage deviation of the best found project duration from the reported upper bound in Coelho and Vanhoucke (2020)[3] (Δ^{ub}), the number of instances for which a better upper bound could be detected ($\#^{\text{ub}}_{<}$), or an equal upper bound has been determined ($\#^{\text{ub}}_{=}$).

The results of the performance analysis on the benchmark sets with less than 100 real activities are indeed promising, where the performance on benchmark set CV should especially be emphasized, for which 27.93% of the instances could be solved to optimality after a maximum run time of 1 h and for which for five instances even better upper bounds could be determined. In contrast, a significant decrease in the performance of BB1 on instances with 100 or more real activities (KSD120, UBO100, UBO200) can be observed, especially with regard to the number of instances for which the solvability status could not be determined. For greater instances of benchmark set UBO with 500 and 1000 real activities (UBO500, UBO1000) BB1 could not be applied due to memory overloads. A closer look on the results over all instances with less than 100 real activities reveals that the scarcity of the resources has the greatest impact on their intractability, while the performance of BB1 is also significantly but less influenced by the average number of the demanded resources over all real activities.

In conclusion, our BnB provides a surprising good performance for small instances with renewable resources, noticing that no instance of CV has been solved by a specified BnB for the RCPSP within 20 h runs (cf. Coelho and Vanhoucke

[3] http://solutionsupdate.ugent.be/solutions-update/results.

2020). However, the experimental performance analysis revealed that our BnB is rather not capable to handle test instances with 100 or more real activities, which can be assumed to be given by the pseudo-polynomial growth in the instance size by the transformation procedure between the resource types.

In our opinion, partially renewable resources should be considered as a concept that makes new types of restrictions amenable to project scheduling, instead of providing a general framework that release from the necessity to use the specific characteristics of different resource types. For this reason, we are convinced that future research should focus on the development of solution procedures that combine methods for partially renewable and classical renewable resources in order to handle problems that cover both resource types. It should be noted that the combination of our BnB with a relaxation-based method for renewable resources as described in Fest et al. (1999), De Reyck and Herroelen (1998), or Schwindt (1998b) seems to be promising for this.

7 Conclusions

We have devised a new enumeration approach for the resource-constrained project duration problem with partially renewable resources and general temporal constraints (RCPSP/max-π) that is based on a stepwise decomposition of the possible resource consumptions by the activities of the project. To improve the performance of the branch-and-bound algorithm (BnB), we have integrated a traversal strategy and consistency tests from the literature, where the enumeration scheme could be identified as the most decisive part of the solution procedure. The results of a comprehensive experimental performance analysis on different benchmark sets could reveal that our exact solution procedure clearly outperforms all other BnB that are available in the open literature so far. Moreover, it could be shown that our BnB also dominates the state-of-the-art heuristics for partially renewable resources on a well known benchmark set. Finally, an outlook on the capability of our BnB to solve instances with renewable resources could demonstrate the limitation of the approach to represent classical renewable by partially renewable resources.

The experimental performance analysis has shown that the current state-of-the-art heuristics for partially renewable resources are clearly outperformed by our new enumeration approach. Therefore, the development of approximation methods for the RCPSP/max-π that are based on our new enumeration scheme seems to be promising. Furthermore, future research should focus on instances that cover both classical renewable and partially renewable resources in order to reduce the gap to real-life projects. For this, as our experimental investigations suggest, it is necessary to combine solution procedures for both resource types to be able to handle instances that are not only interesting from a theoretical point of view. We are convinced that our new enumeration scheme provides a promising starting point for this.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

Ahuja RK, Magnanti TL, Orlin JB (1993) Network flows: theory, algorithms, and applications. Prentice-Hall, Englewood Cliffs

Alvarez-Valdes R, Crespo E, Tamarit JM, Villa F (2006) A scatter search algorithm for project scheduling under partially renewable resources. J Heuristics 12(1):95–113

Alvarez-Valdes R, Crespo E, Tamarit JM, Villa F (2008) GRASP and path relinking for project scheduling under partially renewable resources. Eur J Oper Res 189(3):1153–1170

Alvarez-Valdes R, Tamarit JM, Villa F (2015) Partially renewable resources. In: Schwindt C, Zimmermann J (eds) Handbook on project management and scheduling, vol 1. Springer, Cham, pp 203–227

Androutsopoulos KN, Manousakis EG, Madas MA (2020) Modeling and solving a bi-objective airport slot scheduling problem. Eur J Oper Res 284(1):135–151

Bartsch T, Drexl A, Kröger S (2006) Scheduling the professional soccer leagues of Austria and Germany. Comput Oper Res 33(7):1907–1937

Böttcher J, Drexl A, Kolisch R, Salewski F (1999) Project scheduling under partially renewable resource constraints. Manag Sci 45(4):543–559

Briskorn D, Fliedner M (2012) Packing chained items in aligned bins with applications to container transshipment and project scheduling. Math Methods Oper Res 75(3):305–326

Coelho J, Vanhoucke M (2020) Going to the core of hard resource-constrained project scheduling instances. Comput Oper Res 121

De Reyck B, Herroelen W (1998) A branch-and-bound procedure for the resource-constrained project scheduling problem with generalized precedence relations. Eur J Oper Res 111(1):152–174

Drexl A, Salewski F (1997) Distribution requirements and compactness constraints in school timetabling. Eur J Oper Res 102(1):193–214

Drexl A, Juretzka J, Salewski F (1993) Academic course scheduling under workload and changeover constraints. Working paper 337, University of Kiel

Drexl A, Nissen R, Patterson JH, Salewski F (2000) ProGen/πx - an instance generator for resource-constrained project scheduling problems with partially renewable resources and further extensions. Eur J Oper Res 125(1):59–72

Fest A, Möhring RH, Stork F, Uetz M (1999) Resource-constrained project scheduling with time windows: a branching scheme based on dynamic release dates. Technical Report 596, revised version, Technical University of Berlin

Franck B, Neumann K, Schwindt C (2001) Truncated branch-and-bound, schedule-construction, and schedule-improvement procedures for resource-constrained project scheduling. OR Spektr 23(3):297–324

Knust S (2010) Scheduling non-professional table-tennis leagues. Eur J Oper Res 200(2):358–367

Kolisch R, Sprecher A, Drexl A (1995) Characterization and generation of a general class of resource-constrained project scheduling problems. Manag Sci 41(10):1693–1703

Kolisch R, Schwindt C, Sprecher A (1999) Benchmark instances for project scheduling problems. In: Węglarz J (ed) Project scheduling: recent models, algorithms and applications, vol 14. Kluwer, Boston, pp 197–212

Okubo H, Miyamoto T, Yoshida S, Mori K, Kitamura S, Izui Y (2015) Project scheduling under partially renewable resources and resource consumption during setup operations. Comput Ind Eng 83:91–99

Patterson JH (1984) A comparison of exact approaches for solving the multiple constrained resource, project scheduling problem. Manag Sci 30(7):854–867

Schirmer A (1999) Project scheduling with scarce resources: models, methods, and applications. Kovač, Hamburg

Schirmer A, Drexl A (2001) Allocation of partially renewable resources: concept, capabilities, and applications. Networks 37(1):21–34

Schwindt C (1998a) Generation of resource-constrained project scheduling problems subject to temporal constraints. Report WIOR-543, University of Karlsruhe

Schwindt C (1998b) Verfahren zur Lösung des ressourcenbeschränkten Projektdauerminimierungsproblems mit planungsabhängigen Zeitfenstern (Procedures for the solution of the resource-constrained project duration problem with schedule-dependent time windows). Shaker, Aachen

Talbot FB, Patterson JH (1978) An efficient integer programming algorithm with network cuts for solving resource-constrained scheduling problems. Manag Sci 24(11):1163–1174

Watermeyer K, Zimmermann J (2020a) A branch-and-bound procedure for the resource-constrained project scheduling problem with partially renewable resources and general temporal constraints. OR Spectr 42(2):427–460

Watermeyer K, Zimmermann J (2020b) A constructive branch-and-bound algorithm for the project duration problem with partially renewable resources and general temporal constraints. Manuscript submitted for publication