Watermeyer, Kai; Zimmermann, Jürgen

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

Journal of Scheduling

Provided in Cooperation with:
Springer Nature

Suggested Citation: Watermeyer, Kai; Zimmermann, Jürgen (2022) : A constructive branch-and-bound algorithm for the project duration problem with partially renewable resources and general temporal constraints, Journal of Scheduling, ISSN 1099-1425, Springer US, New York, NY, Vol. 26, Iss. 1, pp. 95-111, https://doi.org/10.1007/s10951-022-00735-9

Journal of Scheduling (2023) 26:95–111
https://doi.org/10.1007/s10951-022-00735-9

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

Kai Watermeyer[1] · **Jürgen Zimmermann**[1]

Accepted: 28 March 2022 / Published online: 6 June 2022

Abstract
This paper deals with the resource-constrained project scheduling problem with partially renewable resources and general temporal constraints with the objective to minimize the project duration. The consideration of partially renewable resources allows to integrate the decision about the availability of a resource for a specific time period into the scheduling process. Together with general temporal constraints, which permit to establish minimum and maximum time lags between activities, even more aspects of real-life projects can be taken into account. We present a branch-and-bound algorithm for the stated problem that is based on a serial schedule-generation scheme. For the first time it is shown how a dominance criterion can be applied on the associated generation scheme to reduce the start times in each scheduling step. To improve the performance of the solution procedure, we integrate consistency tests and lower bounds from the literature and devise new pruning techniques. In a comprehensive experimental performance analysis we compare our exact solution procedure with all available branch-and-bound algorithms for partially renewable resources. Additionally, we investigate a directly derived priority rule-based approximation method from our new enumeration scheme. The results of the computational study demonstrate the efficiency of our branch-and-bound algorithm and reveal that the derived approximation method is only suited to solve small- and medium-sized instances.

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

1 Introduction

In the field of project scheduling, a great deal of effort has been devoted over the years to renewable resources, which are able to model resources like staff or machines that are assumed to be available in a specific quantity at each point in time (or period). In this work, we consider a more general resource type, which has firstly been introduced under the term partially renewable resources in the framework of project scheduling by Böttcher et al. (1999). The corresponding problem, denoted by RCPSP/π, is a generalization of the classical resource-constrained project scheduling problem (RCPSP). The motivation for the extension of RCPSP by partially renewable resources stems from the restrictiveness of the renewable resources in the sense that the availability for each time period has to be fixed in advance, separated from the scheduling process. This limitation is resolved by partially renewable resources that are defined over multiple time periods with a given capacity. Thereby, the resources are consumed only on the defined time periods by the activities of the project. Examples for the application of RCPSP/π can be found in Böttcher et al. (1999) for the flexible planning of lunch breaks, in Alvarez-Valdes et al. (2008) for the assignment of weekend work, or in Alvarez-Valdes et al. (2015) for a school timetabling problem.

In the last decades, approximation and exact solution procedures have been developed for RCPSP/π. In Böttcher et al. (1999) and Schirmer (1999), priority rule-based methods for RCPSP/π are investigated. The works of Alvarez-Valdes et al. (2006, 2008, 2015) are devoted to a GRASP and a

✉ 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

scatter search algorithm, and in Schirmer (1999) different local-search procedures are considered. The state-of-the-art approximation method for RCPSP/π in terms of the solution quality is given by the scatter search algorithm. This procedure makes use of a priority rule-based generation scheme to determine feasible schedules, which are iteratively combined with each other to obtain better solutions. Böttcher et al. (1999) devised the only exact solution procedure for RCPSP/π. The branch-and-bound algorithm (BnB) is based on the enumeration scheme by Talbot and Patterson (1978) for which the authors have developed two feasibility bounds.

For the first time, Watermeyer and Zimmermann (2020) have extended RCPSP/π by taking minimum and maximum time lags between the activities into account (RCPSP/max-π). An example for this problem is the planning of a training program for an employee that consists of a theoretical and a practical course. Thereby, the theory has to be taught first, the practical course is not supposed to be conducted more than two weeks after, and at most one course should take place at a weekend day. The temporal constraints between both courses are represented by minimum and maximum time lags, while the weekend constraint is given by a partially renewable resource that is defined over all weekend days with a capacity for one course. The state-of-the-art exact solution procedure for RCPSP/max-π and RCPSP/π is provided in Watermeyer and Zimmermann (2020) by a relaxation-based BnB. The procedure progressively reduces the possible resource consumptions by the activities of the project in each branching step and solves the corresponding resource relaxations.

In this work we present a new BnB for RCPSP/max-π that is based on a serial schedule-generation scheme. The new enumeration approach schedules all activities of the project successively by assigning a start time to an activity in each branching step. In this way, partial schedules are augmented in the course of the enumeration until there is no feasible start time left or a complete schedule is determined.

The remainder of this paper is organized as follows. Section 2 provides a formal description of RCPSP/max-π. In Sect. 3 we discuss the enumeration scheme of our BnB. Section 4 deals with improving techniques, and Sect. 5 describes the BnB. In Sect. 6 we present the results of a comprehensive experimental performance analysis and provide some conclusions in Sect. 7.

2 Problem description

RCPSP/max-π can be represented by an activity-on-node project network N with node set V, covering all activities of the project, and arc set $E \subset V \times V$, implying the precedence relationships among them. Each activity $i \in V$ is assigned a non-interruptible processing time $p_i \in \mathbb{Z}_{\geq 0}$ and a resource demand $r_{ik}^d \in \mathbb{Z}_{\geq 0}$ for each partially renewable resource $k \in \mathcal{R}$. The temporal constraint for each activity pair $(i, j) \in E$ is specified by a start-to-start precedence relationship and arc weight $\delta_{ij} \in \mathbb{Z}$, meaning that each temporal constraint is given by $S_j \geq S_i + \delta_{ij}$, establishing a time lag between the start times of activities i and j. In the following we speak of a minimum time lag between activities i and j if $\delta_{ij} \geq 0$ and say that a maximum time lag is given if $\delta_{ji} < 0$. The node set $V{:=}\{0, 1, \ldots, n+1\}$ includes two fictitious activities 0 and $n+1$, i.e., $p_0 = p_{n+1} = 0$, which represent the beginning and the termination of the project, respectively. It is assumed that each project starts at time 0 and is completed before a prescribed deadline $\bar{d}$, i.e., $S_0 = 0$ and $S_{n+1} \leq \bar{d}$. In the remainder of this work, we call a vector $S = (S_i)_{i \in V}$ with $S_i \in \mathbb{Z}_{\geq 0}$ and $S_0 = 0$ a schedule and speak of a time-feasible schedule if all temporal constraints are satisfied and $S_{n+1} \leq \bar{d}$. By $\mathcal{S}_T$, we denote the set of all time-feasible schedules. The resource constraints of RCPSP/max-π are given by the resource capacities $R_k \in \mathbb{Z}_{\geq 0}$ of all partially renewable resources $k \in \mathcal{R}$. Thereby, the availability of each resource is only limited on a specified subset of all time periods within the entire planning horizon $\Pi_k \subseteq \{1, 2, \ldots, \bar{d}\}$. As a consequence, only the resource consumption of an activity $i \in V_k{:=}\{i \in V \mid r_{ik}^d > 0\}$ over the time periods in Π_k has to be taken into account. In order to express the number of the time periods in Π_k an activity $i \in V$ is in execution, we introduce the so-called resource usage $r_{ik}^u(S_i){:=}|]S_i, S_i + p_i] \cap \Pi_k|$. Based on the resource usage, the resource consumption of a resource $k \in \mathcal{R}$ by an activity $i \in V$ follows directly with $r_{ik}^c(S_i){:=}r_{ik}^u(S_i) \cdot r_{ik}^d$, so that the resource constraints can be stated by $\sum_{i \in V} r_{ik}^c(S_i) \leq R_k$ for all $k \in \mathcal{R}$. In the following, we call a schedule S that fulfills all resource constraints a resource-feasible schedule and denote the set of all resource-feasible schedules by $\mathcal{S}_R$. Furthermore, we say that schedule $S \in \mathcal{S}$ is feasible with $\mathcal{S}{:=}\mathcal{S}_T \cap \mathcal{S}_R$ as the set of all feasible schedules.

The objective of RCPSP/max-π is to determine a feasible schedule S^* with the shortest project duration among all feasible schedules $S \in \mathcal{S}$. The corresponding problem is stated by

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

with $f : \mathcal{S} \mapsto \mathbb{R}$ as the objective function that assigns the project duration to each feasible schedule S. We call a schedule S that solves problem (P) an optimal schedule and denote the set of all optimal schedules by $\mathcal{OS}$.

It should be noted that there also exist other approaches to model partially renewable resources by assigning multiple subsets of time periods to each of them with the advantage of a more intuitive connection to resources in real-life applications (Böttcher et al. 1999). In this work, we use the so-called normalized formulation for partially renewable resources

that turned out to be more appropriate for theoretical issues since each restriction of a real-life resource is represented by exactly one partially renewable resource.

3 Enumeration scheme

In general, the enumeration scheme of a BnB specifies the procedure to construct the search tree or rather implicates how to generate all direct descendants of a search node. In the following, we present the enumeration scheme of our BnB, which is based on a serial schedule-generation procedure complemented by an unscheduling step. The concept of unscheduling is based on the work of Franck et al. (2001) that provides a serial schedule-generation scheme for RCPSP with general temporal constraints (RCPSP/max).

The construction procedure of the directed outtree corresponding to the enumeration scheme of our BnB is described in Algorithm 1. In this procedure, each enumeration node is represented by a pair $(\mathcal{C}, S)$ with $\mathcal{C} \subseteq V$ as the set of all currently scheduled activities and S as a time-feasible schedule that represents the start times S_i for all activities $i \in \mathcal{C}$ and the earliest time-feasible start times for all not currently scheduled activities $i \in V \setminus \mathcal{C}$. To simplify the following explanations, we use the concept of partial schedules, referring to Definition (2.6.3) in Neumann et al. (2003), to describe the start times of all currently scheduled activities for some enumeration node.

Definition 1 $S^{\mathcal{C}}:=(S_i)_{i\in\mathcal{C}}$ with $\mathcal{C} \subseteq V$ and schedule S is called a partial schedule. In case that $S_j \geq S_i + \delta_{ij}$ for all $(i, j) \in E \cap \mathcal{C} \times \mathcal{C}$, partial schedule $S^{\mathcal{C}}$ is said to be time-feasible and is termed resource-feasible if $\sum_{i\in\mathcal{C}} r^c_{ik}(S_i) \leq R_k$ for all $k \in \mathcal{R}$. In compliance with schedules, a time-feasible and resource-feasible partial schedule $S^{\mathcal{C}}$ is called feasible. $S^{\mathcal{C}\cup\{i\}}$ with $i \in V \setminus \mathcal{C}$ is said to be the augmentation of $S^{\mathcal{C}}$ by activity i and S_i is termed time-feasible, resource-feasible or feasible if partial schedule $S^{\mathcal{C}\cup\{i\}}$ is time-feasible, resource-feasible or feasible, respectively.

Algorithm 1 outlines the enumeration scheme of our BnB. In the initialization step, the distance matrix $D:=(d_{ij})_{i,j\in V}$ with d_{ij} as the length of a longest path between activities i and j in project network N is calculated with the Floyd-Warshall algorithm (Ahuja et al. 1993, Sect. 5.6). Then, the earliest and latest time-feasible schedules ES and LS are derived and the root node $(\mathcal{C}, S)$ is initialized by $\mathcal{C}:=\{0\}$ and $S:=ES$, meaning that the project start is scheduled at time 0, so that partial schedule $S^{\mathcal{C}} = (0)$ with $S_0 = 0$ corresponds to the root node. For the enumeration scheme we use set Ω to store all generated enumeration nodes that are still to be explored, and Φ to gather all generated schedules during the construction procedure. Accordingly, these sets are initialized by $\Omega:=\{(\mathcal{C}, S)\}$ and $\Phi:=\emptyset$.

Algorithm 1: Enumeration scheme

Input: Instance of problem RCPSP/max-π
Output: Set Φ of candidate schedules

```
1  Determine distance matrix D = (d_ij)_{i,j∈V}
2  Set ES_i := d_0i, LS_i := −d_i0 for all i ∈ V
3  C := {0}   S := ES
4  Ω := {(C, S)}   Φ := ∅
5  while Ω ≠ ∅ do
6      Remove (C, S) from set Ω
7      if C = V then
8          Φ := Φ ∪ {S}
9      else
10         Select activity i ∈ C̄
11         Θ_i := {τ ∈ {S_i, ..., LS_i} | r^c_k(S^C) + r^c_ik(τ) ≤ R_k for all
           k ∈ R_i}
12         Calculate T_i (Algorithm 2)
13         forall the t ∈ T_i do
14             S'_i := t   S' := (max(S_j, S'_i + d_ij))_{j∈V}
15             if ∃j ∈ C : S'_j > S_j then
16                 C' := C \ {j ∈ C | S'_j > S_j}
17             else
18                 C' := C ∪ {i}
19             Ω := Ω ∪ {(C', S')}
20 return Φ
```

In each iteration of Algorithm 1 some pair $(\mathcal{C}, S)$ is removed from Ω. In case that $\mathcal{C} \neq V$, some activity $i \in \bar{\mathcal{C}}$ from the set of all not currently scheduled activities $\bar{\mathcal{C}}:=V \setminus \mathcal{C}$ is chosen. For this activity, all resource-feasible start times in $\{S_i, \ldots, LS_i\}$ are determined and stored in the so-called scheduling set Θ_i. The resource-feasibility is ensured by taking the resource consumption $r^c_k(S^{\mathcal{C}}):=\sum_{j\in\mathcal{C}} r^c_{jk}(S_j)$ for each resource $k \in \mathcal{R}_i:=\{k \in \mathcal{R} \mid r^d_{ik} > 0\}$ of partial schedule $S^{\mathcal{C}}$ into account. Afterward, based on a dominance criterion with similarities to left-shift dominance rules for renewable resources (see, e.g., Stinson et al., 1978; Demeulemeester and Herroelen, 1992), the reduced scheduling set T_i is calculated. T_i comprises all start times from Θ_i with a lower resource consumption for at least one resource $k \in \mathcal{R}_i$ compared to all lower start times in Θ_i. The calculation of T_i, which is defined by $T_i:=\{\tau\in\Theta_i \mid \nexists\tau'\in[0, \tau[\cap\Theta_i : r^u_{ik}(\tau) \geq r^u_{ik}(\tau')$ for all $k \in \mathcal{R}_i\}$, is described in Sect. 3.1. Based on the reduced scheduling set T_i, all direct descendants of enumeration node $(\mathcal{C}, S)$ are generated. For each descendant node $(\mathcal{C}', S')$, some scheduling time $t \in T_i$ is established as the start time S'_i of activity i, followed by an update of the earliest time-feasible start times for all activities by $S':=(\max(S_j, S'_i + d_{ij}))_{j\in V}$ if it is assumed that activity i starts at time t. Since for each start time $t \in T_i$ the conditions $S_j + d_{ji} \leq t$ for all currently scheduled activities $j \in \mathcal{C}$ are satisfied, in case that t is not time-feasible, there has to be at least one activity $j \in \mathcal{C}$ with $S_j - d_{ij} < t$. This means that the induced latest start time $LS_i(S_j):=S_j - d_{ij}$ of activity i by some activity $j \in \mathcal{C}$ prevents the time-feasibility of start

time t. As a consequence, in order to achieve the time-feasibility of $S'^{\mathcal{C}\cup\{i\}}$ with $S'_i = t$, all currently scheduled activities $j \in \mathcal{C}$ with $t > LS_i(S_j)$ or rather $S'_j > S_j$ are unscheduled by setting $\mathcal{C}' := \mathcal{C} \setminus \{j \in \mathcal{C} \mid S'_j > S_j\}$. More precisely, the unscheduling of an activity is done by removing it from the set of all currently scheduled activities $\mathcal{C}$ of the direct ancestor node. It should be noted that $0 \in \mathcal{C}'$ is always ensured by $t \leq LS_i$ due to the definition of Θ_i. In case that start time t is time-feasible ($t \leq \min_{j\in\mathcal{C}} LS_i(S_j)$), activity i is scheduled at start time $S'_i = t$ by setting $\mathcal{C}' := \mathcal{C} \cup \{i\}$. Finally, after the scheduling or unscheduling step, the descendant node $(\mathcal{C}', S')$ is stored in Ω in order to be explored in one of the following iterations. From the description of the procedure to schedule or unschedule activities it follows directly that each partial schedule $S^{\mathcal{C}}$ of an enumeration node $(\mathcal{C}, S)$ is feasible. Therefore, in case that some node $(\mathcal{C}, S)$ with $\mathcal{C} = V$ is removed from Ω, schedule $S = S^{\mathcal{C}} \in \mathcal{S}$ is stored in Φ as a candidate schedule. After all enumeration nodes have been explored, i.e., $\Omega = \emptyset$, Algorithm 1 terminates and returns set Φ, which contains all candidate schedules generated in the course of the enumeration procedure.

3.1 Reduced scheduling set

This section deals with the calculation of the reduced scheduling set T_i in each branching step of Algorithm 1. For the first time, we provide a procedure to determine dominant start times from Θ_i, while all existing schedule-generation schemes for partially renewable resources consider all start times from Θ_i in each scheduling step (see Schirmer, 1999; Alvarez-Valdes et al., 2006; 2008; 2015).

Algorithm 2: Reduced scheduling set

Input: Scheduling set Θ_i
Output: Reduced scheduling set T_i

```
1  T_i := ∅   t := min Θ_i   element := true
2  while t < ∞ do
3  |  forall the τ ∈ T_i do
4  |  |  element := false
5  |  |  forall the k ∈ R_i do
6  |  |  |  if r^u_ik(t) < r^u_ik(τ) then
7  |  |  |  |  element := true
8  |  |  |  |  break
9  |  |  if element = false then
10 |  |  |  break
11 |  if element = true then
12 |  |  T_i := T_i ∪ {t}
13 |  t := min{τ ∈ Θ_i | τ > t}
14 return T_i
```

The procedure to calculate the reduced scheduling set $T_i := \{\tau \in \Theta_i \mid \nexists \tau' \in [0, \tau[[0] \cap \Theta_i : r^u_{ik}(\tau) \geq r^u_{ik}(\tau')$ for all $k \in \mathcal{R}_i\}$ from Θ_i is given in Algorithm 2. To simplify the representation, the definition $\min \emptyset := \infty$ is assumed. In the course of the procedure, variable t serves as the start time that is considered in the current iteration, while boolean variable *element* is used to indicate if start time t is or is not an element of T_i. The algorithm starts with an empty set T_i and checks in each iteration for some start time $t \in \Theta_i$ if there is any start time $\tau \in T_i$ with $r^u_{ik}(t) \geq r^u_{ik}(\tau)$ for all $k \in \mathcal{R}_i$. This is done in the procedure by checking for each $\tau \in T_i$ if there is at least one resource $k \in \mathcal{R}_i$ with $r^u_{ik}(t) < r^u_{ik}(\tau)$. If this is the case for all start times in T_i, meaning that there is no start time $\tau \in T_i$ with $r^u_{ik}(t) \geq r^u_{ik}(\tau)$ for all $k \in \mathcal{R}_i$, t is added to T_i. Since all start times $t \in \Theta_i$ are considered in an increasing order, the condition that there is no start time $\tau \in T_i$ with $r^u_{ik}(t) \geq r^u_{ik}(\tau)$ for all $k \in \mathcal{R}_i$ implies that there is also no lower start time in Θ_i satisfying this condition. This can be verified by noticing that for each start time $\tau \in \Theta_i$ that has not been added to T_i in the course of Algorithm 2, there is at least one lower start time $\tau' \in T_i$ with $r^u_{ik}(\tau') \leq r^u_{ik}(\tau)$ for all $k \in \mathcal{R}_i$. Finally, since for each start time $t \in \Theta_i$ that is not an element of T_i at the end of the procedure there is at least one earlier start time $\tau \in \Theta_i$ ($\tau \in T_i$) with $r^u_{ik}(t) \geq r^u_{ik}(\tau)$ for all $k \in \mathcal{R}_i$, we can state that Algorithm 2 is correct.

3.2 Correctness of the enumeration scheme

In what follows, we prove that Algorithm 1 generates at least one optimal schedule in finitely many iterations if and only if there is at least one feasible solution. It should be noted that the total correctness of Algorithm 1 follows directly from this proof since each candidate schedule is feasible, i.e., $\Phi \subseteq \mathcal{S}$. First of all, Theorem 1, which is based on Lemmas 1 and 2, states that the enumeration scheme generates at least the set of all so-called active schedules $\mathcal{AS}$. In line with Neumann et al. (2000), we call a feasible schedule S active if and only if there is no feasible schedule $S' \neq S$ with $S' \leq S$, i.e., $S'_i \leq S_i$ for all $i \in V$. By definition, for each non-active feasible schedule there is at least one activity whose start time can be left-shifted to obtain a feasible schedule. As a consequence, there has to be at least one optimal schedule that is active for each instance with $\mathcal{S} \neq \emptyset$, so that Theorem 1 implicates the completeness of Algorithm 1, i.e., $\mathcal{S} \neq \emptyset \Leftrightarrow \Phi \cap \mathcal{OS} \neq \emptyset$.

Lemma 1 *Let T_i be the reduced scheduling set of Θ_i. Then T_i contains exactly all lowest scheduling times $t \in \Theta_i$ that satisfy $r^u_{ik}(t) \leq r^u_{ik}(\tau)$ for any $\tau \in \Theta_i$ and all $k \in \mathcal{R}_i$, i.e., $T_i = T_i^{\cup} := \bigcup_{\tau\in\Theta_i} \{\min\{\tau' \in \Theta_i \mid r^u_{ik}(\tau') \leq r^u_{ik}(\tau)$ for all $k \in \mathcal{R}_i\}\}$.*

Proof Consider any start time $t \in T_i$. From the definition of T_i it follows directly that $t = \min\{\tau \in \Theta_i \mid r^u_{ik}(\tau) \leq r^u_{ik}(t)$ for all $k \in \mathcal{R}_i\}$, so that $T_i \subseteq T_i^{\cup}$ is given. Next, let $\tau \in \Theta_i$ and $t := \min\{\tau' \in \Theta_i \mid r^u_{ik}(\tau') \leq r^u_{ik}(\tau)$ for all $k \in \mathcal{R}_i\}$ be given and assume $t \notin T_i$. Since $t \notin T_i$ implies $t >$

$\min\{\tau' \in \Theta_i \mid r^u_{ik}(\tau') \leq r^u_{ik}(\tau) \text{ for all } k \in \mathcal{R}_i\}$, which would contradict the assumption for t, $T_i \supseteq T_i^{\cup}$ and therefore $T_i = T_i^{\cup}$ follows. □

Lemma 2 *Let $S^f \in \mathcal{S}$ be any feasible schedule and $(\mathcal{C}, S)$ some node corresponding to the enumeration scheme of Algorithm 1 with $\mathcal{C} \neq V$, $S \leq S^f$ and $r^u_{jk}(S_j) \leq r^u_{jk}(S^f_j)$ for all $j \in \mathcal{C}$ and all $k \in \mathcal{R}_j$. Then there is at least one direct descendant node $(\mathcal{C}', S')$ that fulfills the conditions $S' \leq S^f$ and $r^u_{jk}(S'_j) \leq r^u_{jk}(S^f_j)$ for all $j \in \mathcal{C}'$ and all $k \in \mathcal{R}_j$.*

Proof Let $i \in \bar{\mathcal{C}}$ be the selected activity for the generation of the direct descendants of enumeration node $(\mathcal{C}, S)$. First of all, $S^f_i \in \Theta_i$ can be derived from $S_i \leq S^f_i \leq LS_i$ and $r^c_k(S^{\mathcal{C}}) + r^c_{ik}(S^f_i) \leq r^c_k(S^f) \leq R_k$ for all $k \in \mathcal{R}_i$. Since $S^f_i \in \Theta_i$, from Lemma 1 we get $t := \min\{\tau \in \Theta_i \mid r^u_{ik}(\tau) \leq r^u_{ik}(S^f_i) \text{ for all } k \in \mathcal{R}_i\} \in T_i$, so that $t \leq S^f_i$ and $r^u_{ik}(t) \leq r^u_{ik}(S^f_i)$ for all $k \in \mathcal{R}_i$. Accordingly, considering the direct descendant node corresponding to start time t of activity i, $S' \leq S^f$ is implied by $t \leq S^f_i$ $(t + d_{ij} \leq S^f_i + d_{ij} \leq S^f_j$ for all $j \in V)$ and the conditions $r^u_{jk}(S'_j) \leq r^u_{jk}(S^f_j)$ for all $j \in \mathcal{C}'$ and $k \in \mathcal{R}_j$ are satisfied as well, either if activity i is scheduled $(\mathcal{C}' := \mathcal{C} \cup \{i\})$ or some activities are unscheduled. □

Theorem 1 *Algorithm 1 generates all active schedules, i.e., $\Phi \supseteq \mathcal{AS}$.*

Proof For each active schedule $S^a \in \mathcal{AS}$ the conditions $S \leq S^a$ and $r^u_{jk}(S_j) \leq r^u_{jk}(S^a_j)$ for all $j \in \mathcal{C}$ and all $k \in \mathcal{R}_j$ are satisfied with $(\mathcal{C}, S)$ corresponding to the root node. Accordingly, it follows from Lemma 2 that there exists at least one path in the enumeration tree on which each node $(\mathcal{C}, S)$ satisfies the conditions $S \leq S^a$ and $r^u_{jk}(S_j) \leq r^u_{jk}(S^a_j)$ for all $j \in \mathcal{C}$ and all $k \in \mathcal{R}_j$, respectively. Since for the generation of any direct descendant node either the start time S_j for at least one activity $j \in \mathcal{C}$ is increased $(S'_j > S_j)$ or some activity $i \in \bar{\mathcal{C}}$ is scheduled $(\mathcal{C}' := \mathcal{C} \cup \{i\})$, each such path has a finite length. Finally, from the property of Algorithm 1 that each generated schedule $S \in \Phi$ is feasible and since $S \leq S^a$ with $S \neq S^a$ would contradict the assumption that S^a is active, we can state that Algorithm 1 generates all active schedules. □

Finally, Lemma 3 establishes that the enumeration scheme terminates after a finite number of iterations.

Lemma 3 *Algorithm 1 generates at most $(\bar{d}+1)^{|V|(\bar{d}+1)}$ enumeration nodes.*

Proof For the generation of any descendant node in the enumeration scheme of Algorithm 1 either the selected activity $i \in \bar{\mathcal{C}}$ is scheduled or the start time S_j of any activity $j \in \mathcal{C}$ is increased by at least one unit. Since the start time of each activity is bounded from above by $\bar{d} + 1$, an upper bound for the maximum depth of the enumeration tree is given by $|V|(\bar{d} + 1)$. Accordingly, an upper bound for the maximum number of generated nodes is given by $(\bar{d} + 1)^{|V|(\bar{d}+1)}$ taking into consideration that the number of start times in T_i is bounded from above by $\bar{d} + 1$. □

4 Improving techniques

In Watermeyer and Zimmermann (2020) it has already been shown for a relaxation-based BnB for RCPSP/max-π that the application of consistency tests, lower bounds , and techniques to avoid redundancies can have a great impact on the performance. In what follows, we extend our enumeration scheme to be able to use consistency tests and lower bounds from Watermeyer and Zimmermann (2020) and to apply new devised pruning techniques that are described in Sect. 4.3.

4.1 Extended enumeration scheme

For the extension of the enumeration scheme we establish a domain $W_i \subseteq H := \{0, 1, \ldots, \bar{d}\}$ for the start time S_i of each activity $i \in V$. Set W_i contains all possible start times of activity $i \in V$, i.e., $S_i \in W_i$. In line with Definition 1 in Watermeyer and Zimmermann (2020), we call $W := (W_i)_{i \in V}$ with $W_i \subseteq H$ for all $i \in V$ and $W_0 = \{0\}$ a start time restriction and denote by W_i the start time restriction of activity $i \in V$. In the following we speak of a W-feasible schedule S if $S \in \mathcal{S}_T(W) := \{S \in \mathcal{S}_T \mid S_i \in W_i \text{ for all } i \in V\}$ and say that a partial schedule $S^{\mathcal{C}}$ is W-feasible if there is at least one schedule $S' \in \mathcal{S}_T(W)$ with $S'_i = S_i$ for all scheduled activities $i \in \mathcal{C}$. Furthermore, in accordance with Definition 1, we say that start time S_i of some not currently scheduled activity $i \in \bar{\mathcal{C}}$ is W-feasible exactly if augmentation $S^{\mathcal{C} \cup \{i\}}$ is W-feasible. Therefore, if t is established as the earliest possible start time of some activity $i \in V$, the earliest W-feasible start time of any activity $j \in V$ is expressed by $ES_j(W, i, t) := (\min \widetilde{\mathcal{S}}_T(W, i, t))_j$ with $\widetilde{\mathcal{S}}_T(W, i, t) := \{S \in \mathcal{S}_T(W) \mid S_i \geq t\}$. In the same way, the latest W-feasible start time of an activity $j \in V$ is given by $LS_j(W, i, t) := (\max \widehat{\mathcal{S}}_T(W, i, t))_j$ with $\widehat{\mathcal{S}}_T(W, i, t) := \{S \in \mathcal{S}_T(W) \mid S_i \leq t\}$ if t is assumed to be the latest possible start time of activity $i \in V$. In Watermeyer and Zimmermann (2020), two algorithms have been introduced that are able to determine the minimal point of $\widetilde{\mathcal{S}}_T(W, i, t)$ and the maximal point of $\widehat{\mathcal{S}}_T(W, i, t)$, respectively.

The extension of the enumeration scheme is given in Algorithm 3. For each enumeration node a start time restriction W is stored in addition, so that each node is given by a triple $(\mathcal{C}, S, W)$. At the beginning of the algorithm, $W_i := \{ES_i, \ldots, LS_i\}$ for all $i \in V$ ensures that all feasible schedules $S \in \mathcal{S}$ are covered by the set of all W-feasible

Algorithm 3: Extended enumeration scheme

Input: Instance of problem RCPSP/max-π
Output: Set Φ of candidate schedules

Determine distance matrix $D = (d_{ij})_{i,j \in V}$
$ES_i := d_{0i}$, $LS_i := -d_{i0}$ for all $i \in V$
$W_i := \{ES_i, \ldots, LS_i\}$ for all $i \in V$
$\mathcal{C} := \{0\}$ $S := ES$
$\Omega := \{(\mathcal{C}, S, W)\}$ $\Phi := \emptyset$
while $\Omega \neq \emptyset$ **do**
  Remove $(\mathcal{C}, S, W)$ from set Ω
  if $\mathcal{C} = V$ **then**
    $\Phi := \Phi \cup \{S\}$
  else
    Select activity $i \in \bar{\mathcal{C}}$
    $\Theta_i := \{\tau \in W_i \mid r^c_k(S^{\mathcal{C}}) + r^c_{ik}(\tau) \leq R_k \text{ for all } k \in \mathcal{R}_i\}$
    Calculate T_i (Algorithm 2)
    forall the $t \in T_i$ **do**
      $S'_i := t$ $S' := \min \widetilde{\mathcal{S}}_T(W, i, S'_i)$
      $W' := (W_j \setminus [0, S'_j[)_{j \in V}$
      if $\exists j \in \mathcal{C} : S'_j > S_j$ **then**
        $\mathcal{C}' := \mathcal{C} \setminus \{j \in \mathcal{C} \mid S'_j > S_j\}$
      else if $S'_i = t$ **then**
        $\mathcal{C}' := \mathcal{C} \cup \{i\}$
      $\Omega := \Omega \cup \{(\mathcal{C}', S', W')\}$
return Φ

schedules in the root node, i.e., $\mathcal{S}_T(W) \supseteq \mathcal{S}$. In the further course of the algorithm, recalling that W_i represents the domain of start time S_i, all start times in T_i of some activity $i \in \bar{\mathcal{C}}$ are limited to W_i. Accordingly, each start time $t \in T_i$ that is assigned to activity $i \in \bar{\mathcal{C}}$ is assured to be an element of W_i. In the branching step, in order to generate a descendant node $(\mathcal{C}', S', W')$, some start time $t \in T_i$ is established as the earliest start time of activity $i \in \bar{\mathcal{C}}$ by setting $S'_i := t$. Following, the earliest W-feasible schedule S' with $S' \geq S$ and $S'_i \geq t$ is determined. Since $S = (\min W_i)_{i \in V}$ is assured at the start of each iteration, the earliest W-feasible schedule with $S' \geq S$ and $S'_i \geq t$ is obtained by $S' := \min \widetilde{\mathcal{S}}_T(W, i, t)$. If there is at least one scheduled activity $j \in \mathcal{C}$ with $ES_j(W, i, t) > S_j$ $(S'_j > S_j)$, which implies that $t > LS_i(W, j, S_j)$, start time t of activity i is not W-feasible. As a consequence, all activities $j \in \mathcal{C}$ with $S'_j > S_j$ have to be unscheduled, so that activity i can be scheduled W-feasible at start time t in case that $t = ES_i(W, i, t)$ $(S'_i = t)$. It should be noted that in general, due to the breaks in W, $S'_i = t$ is not assured even if there is no scheduled activity $j \in \mathcal{C}$ with $S'_j > S_j$. Accordingly, it is only possible to schedule activity i W-feasible at time t if $S'_i = t$.

The proof of the total correctness of the extended enumeration scheme is closely related to that of Algorithm 1. In Lemma 2, only the conditions $S^f \in \mathcal{S}_T(W)$ for node $(\mathcal{C}, S, W)$ and $S^f \in \mathcal{S}_T(W')$ for its direct descendant node $(\mathcal{C}', S', W')$ have to be considered in addition. Based on this, the completeness of Algorithm 3 follows analogously to the proof of Theorem 1, while Lemma 3, which still applies to Algorithm 3, states the total correctness.

4.2 Lower bounds and consistency tests

In this section, we shortly present lower bounds and consistency tests that have been developed in Watermeyer and Zimmermann (2020).

The first lower bound is equal to the earliest time-feasible project termination if the start time restrictions of all activities are taken into account. The corresponding lower bound is given by $LB0^{\pi} := ES_{n+1}(W)$ with $ES(W) := \min \mathcal{S}_T(W)$. The second lower bound LBD^{π} is determined in a destructive way, meaning that a hypothetical upper bound on the project duration d is increased as long as it precludes any feasible solution (Brucker and Knust, 2003). To determine the destructive lower bound LBD^{π} in any node $(\mathcal{C}, S, W)$, a binary search is conducted on some time interval $[LB0^{\pi}, UB - 1]$ with UB as the best solution that has been found in the course of the BnB or $\bar{d} + 1$, otherwise. In each iteration, it is checked if the sum of the minimum possible consumptions $\min\{r^c_{ik}(\tau) \mid \tau \in W_i \cap [ES_i(W), LS_i(W, n+1, d)]\}$ over all activities exceeds the capacity of at least one resource k. In this case, there cannot be any feasible schedule with $S_{n+1} \leq d$, so that $d + 1$ is established as a lower bound for the binary search. Otherwise, $d - 1$ is set as an upper bound. This procedure continues until LBD^{π} is determined, i.e., the lowest hypothetical upper bound on the project duration that does not preclude any feasible schedule.

In general, a consistency test establishes an implicit constraint of a problem if some specified condition is satisfied. For all consistency tests we consider, these implicit constraints are unary on the start time of some activity. Accordingly, each consistency test is described by a condition and a reduction rule on the start time restriction of some activity. In line with Dorndorf et al. (2000a), each of the following consistency tests can be interpreted as a function γ mapping any start time restriction W to an updated start time restriction $W' := \gamma(W)$ with $W'_i \subseteq W_i$ for all $i \in V$.

The first three consistency tests are based on the temporal constraints $S_j \geq S_i + \delta_{ij}$ for all $(i, j) \in E$ of problem (P), so that they could be applied on any project scheduling problem independent on the considered resource type. The following two consistency tests are well known and have already been applied on project scheduling problems (see, e.g., Dorndorfet al., 2000b; Alvarez-Valdes et al., 2008). The first (second) test is based on checking for some activity pair $(i, j) \in E$ if the currently lowest (greatest) possible start time $\underline{W}_j := \min W_j$ $(\overline{W}_i := \max W_i)$ of activity j (i) is consistent with the lowest (greatest) possible start time $\min W_i$ $(\max W_j)$ of activity i (j) with respect to time lag δ_{ij}. The corresponding conditions and reduction rules are given as follows:

$$\underline{W}_j < \underline{W}_i + \delta_{ij} \quad \Rightarrow \quad W_j := W_j \setminus [0, \underline{W}_i + \delta_{ij}[$$
$$\overline{W}_i > \overline{W}_j - \delta_{ij} \quad \Rightarrow \quad W_i := W_i \setminus]\overline{W}_j - \delta_{ij}, \infty[$$

In this work, both tests are gathered under the term temporal-bound consistency test.

The next consistency test, which is also based on the temporal constraints of problem (P), checks for each possible start time of some activity whether there even exists any W-feasible schedule with this start time of the activity. One pass of the so-called temporal consistency test considers all start times in the start time restrictions over all activities. The corresponding condition and reduction rule for some start time $t \in W_i$ of an activity $i \in V$ is given by

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

Next, we deal with consistency tests that take the resource constraints of problem (P) into account. All the following tests have in common that they check for each start time of some activity if its induced resource consumption and the minimum consumptions over all other activities exceed the capacity of at least one resource. The general condition is stated by

$$\exists k \in \mathcal{R} : r^c_{ik}(t) + \sum_{j \in V_k \setminus \{i\}} \min\{r^c_{jk}(\tau) \mid \tau \in W^r_j\} > R_k.$$

The difference between the consistency tests is given by the determination of the possible start times W^r_j for the activities $j \in V \setminus \{i\}$. For the resource-bound consistency test, W^r_j is given by the start time restriction, i.e., $W^r_j := W_j$. Extensions of the resource-bound consistency test are the so-called D-interval and W-interval consistency tests that restrict the number of start times in W^r_j by taking temporal constraints into account. While the D-interval consistency test considers $W^r_j := W_j \cap [t + d_{ij}, t - d_{ji}]$, the W-interval consistency test restricts W^r_j even more by setting $W^r_j := W_j \cap [ES_j(W, i, t), LS_j(W, i, t)]$.

4.3 Pruning the enumeration tree

In this section, we present two techniques that are able to reduce the set of schedules that are explored by a node in the course of the enumeration scheme. For both methods, the branching step of Algorithm 3 is extended by a procedure that reduces the start time domain of the branching activity. It should be noted that in contrast to methods that remove generated nodes, the following techniques prune parts of the enumeration tree by adding constraints to each node in the branching step to prevent that parts of the search tree are generated at all. In a first step we develop the two pruning techniques separately from each other. Afterward, we show that the combination of both methods ensures that each part of the time-feasible region is explored exactly once in the course of the enumeration scheme.

The first technique is based on the storage of the resource usage that is induced by the selected activity and its start time in the branching step as a lower bound for all possible start times of the considered activity. In order to apply the so-called usage-preserving technique (UPT), the branching step of the extended enumeration scheme is adapted as follows. Before start time $t \in T_i$ is established as the earliest start time of some activity $i \in \bar{\mathcal{C}}$ in line 15 of Algorithm 3, $W' := W$ is initialized and $W'_i := \{\tau \in W'_i \mid r^u_{ik}(\tau) \geq r^u_{ik}(t) \text{ for all } k \in \mathcal{R}_i\}$ is set. Additionally, for all further operations in line 15, start time restriction W is replaced by W'. Since the application of UPT in Algorithm 3 implies for any node $(\mathcal{C}, S, W)$ and any schedule $S' \in \mathcal{S}_T(W)$ that the conditions $S \leq S'$ and $r^u_{jk}(S_j) \leq r^u_{jk}(S'_j)$ for all $j \in \mathcal{C}$ and all $k \in \mathcal{R}_j$ are satisfied, Corollary 1 follows.

Corollary 1 *Let UPT be used in Algorithm 3, let $S^f \in \mathcal{S}$ be any feasible schedule, and $(\mathcal{C}, S, W)$ some node corresponding to the enumeration scheme of Algorithm 3 with $\mathcal{C} \neq V$ and $S^f \in \mathcal{S}_T(W)$. Then there is at least one direct descendant node $(\mathcal{C}', S', W')$ that fulfills the condition $S^f \in \mathcal{S}_T(W')$.*

As a consequence of Corollary 1, noticing that $\mathcal{S}_T(W') \subseteq \mathcal{S}_T(W)$ for some node $(\mathcal{C}, S, W)$ and any of its descendants $(\mathcal{C}', S', W')$, precisely $\mathcal{S}_T(W)$ is completely explored by enumeration node $(\mathcal{C}, S, W)$ and all its descendants. In other words, there is no descendant node that explores a part of the time-feasible region $\mathcal{S}_T$ that is not a part of $\mathcal{S}_T(W)$. It should be noted that due to the unscheduling of activities this is not assured if UPT is not applied.

The second pruning technique is based on the consideration of the resource usages of all start times in the reduced scheduling set that are lower than the established earliest start time of the selected activity in the branching step. To apply the second pruning technique, the same extensions as for the first method are made in line 15 of Algorithm 3, except for the setting of W'_i that is replaced by $W'_i := \{\tau \in W'_i \mid \nexists \tau' \in [0, t[\cap T_i : r^u_{ik}(\tau) \geq r^u_{ik}(\tau') \text{ for all } k \in \mathcal{R}_i\}$. Accordingly, the second method removes each start time τ from W'_i if there is at least one start time τ' in the reduced scheduling set T_i that is lower than t and satisfies $r^u_{ik}(\tau) \geq r^u_{ik}(\tau')$ for all $k \in \mathcal{R}_i$. Since this implies for each start time $\tau \in W'_i$ that there is at least one resource $k \in \mathcal{R}_i$ with $r^u_{ik}(\tau) < r^u_{ik}(\tau')$ for each $\tau' \in [0, t[\cap T_i$, we call this method usage-limitation technique (ULT).

In what follows, we investigate the application of both pruning techniques in Algorithm 3. For this, we consider two direct descendants $(\mathcal{C}', S', W')$ and $(\mathcal{C}'', S'', W'')$ of some node in the enumeration tree. In addition, we assume that both nodes $(\mathcal{C}', S', W')$ and $(\mathcal{C}'', S'', W'')$ are generated by establishing t' and t'' with $t' < t''$ as the earliest start time of the branching activity $i \in \bar{\mathcal{C}}$, respectively. Considering

the specifications of both pruning techniques to reduce the start time restriction of the branching activity, $W_i' \cap W_i'' = \emptyset$ is given. Since $W_i' \cap W_i'' = \emptyset$ directly implies $\mathcal{S}_T(W') \cap \mathcal{S}_T(W'') = \emptyset$, Corollary 2 follows from Corollary 1.

Corollary 2 *Let UPT and ULT be used in Algorithm 3, let $S^f \in \mathcal{S}$ be any feasible schedule, and $(\mathcal{C}, S, W)$ some node corresponding to the enumeration scheme of Algorithm 3 with $\mathcal{C} \neq V$ and $S^f \in \mathcal{S}_T(W)$. Then there is exactly one direct descendant node $(\mathcal{C}', S', W')$ that fulfills the condition $S^f \in \mathcal{S}_T(W')$.*

Taking $\mathcal{S}_T(W') \subseteq \mathcal{S}_T(W)$ for some node $(\mathcal{C}, S, W)$ and any of its descendants $(\mathcal{C}', S', W')$ into account, from Corollary 2 it follows that each candidate schedule is generated exactly once if both pruning techniques UPT and ULT are used. Furthermore, we can state that in the course of the enumeration, each part of the time-feasible region is explored exactly once, so that any redundancy is excluded.

5 Branch-and-bound algorithm

In this section we present the general framework of our BnB that enables a wide range of different settings concerned with the construction of the enumeration tree and the application of improving techniques. The first part of this section is devoted to the search strategy of our BnB that determines the way to construct the enumeration tree. In order to provide a generic framework for the construction of the search tree, in line with Watermeyer and Zimmermann (2020), we divide the search strategy in different parts, called traversing, branching, generation, and ordering strategy.

For the traversing strategy, which determines the node to be considered next in the course of the BnB, we have implemented two alternatives. The first alternative is the well-known depth-first search (DFS), while the second one is an extension of the DFS that has been introduced by Watermeyer and Zimmermann (2020). The so-called scattered-path search (SPS) selects after a predefined time span, among all not completely explored nodes with lowest search tree level, a node with the lowest bound on the project duration. After some node has been chosen to be explored next, the branching strategy determines the activity to be considered in the branching step. For this, in a first step, the so-called eligible set $\mathcal{E} \subseteq \bar{\mathcal{C}}$, i.e., the set of all activities that could be used for the branching step, is determined. The first alternative takes all not currently scheduled activities into consideration ($\bar{\mathcal{C}}$), i.e., $\mathcal{E} := \bar{\mathcal{C}}$. In contrast, the other two alternatives that are both based on a strict order $\prec$ in set V, reduce the set $\bar{\mathcal{C}}$, where $\mathcal{E} \subseteq \{i \in \bar{\mathcal{C}} \mid Pred^{\prec}(i) \subseteq \mathcal{C}\}$ holds with $Pred^{\prec}(i)$ as the set of all direct predecessors of activity $i \in V$ in a precedence graph $G^{\prec}$ with V as the node set and the covering relation $cr(\prec)$ of the strict order as the arc set. The strict orders we use in this work have been introduced as distance order ($\prec_{\mathrm{D}}$) and cycle order ($\prec_{\mathrm{C}}$) in Franck et al. (2001) and Neumann et al. (2003, Sect. 2.6). For further details we refer the reader to those references. In the following, we assume that the eligible set $\mathcal{E}$ is given. Then in the next step of the branching strategy, the activity with the best priority value π_i and the lowest index in set $\mathcal{E}$ is selected for the branching step. Accordingly the branching activity is given by

$$i := \min\{i' \in \mathcal{E} \mid \pi_{i'} = \operatorname*{ext}_{h \in \mathcal{E}} \pi_h\},$$

where ext $\in$ {min, max} indicates if lower (min) or greater (max) priority values are preferred. In what follows, we present some priority rules that have shown promising results in preliminary tests. First, we deal with priority rules that have already been discussed in the literature (see, e.g., Kolisch, 1996; Franck et al., 2001) and are concerned with the temporal constraints of the problem. These include among others the latest start time rule (LST) with $\pi_i = LS_i$ and the slack time rule (ST) with $\pi_i = LS_i - ES_i$. For both priority rules we have also tested dynamic versions that take the start time restrictions and the best found solution UB into account that are given by $\pi_i = LS_i^{UB}(W)$ (LSTd) and $\pi_i = LS_i^{UB}(W) - ES_i(W)$ (STd) with $LS_i^{UB}(W) := LS_i(W, n+1, UB-1)$. Additionally, we have implemented a dynamic version of ST (STd$^{\mathrm{I}}$) that considers the number of start times in the start time restriction by $\pi_i = |W_i \cap [ES_i(W), LS_i^{UB}(W)]|$. Further rules are given by the total successor rule (TS) with $\pi_i = |\mathrm{Reach}^{\prec}(i)|$ and $\mathrm{Reach}^{\prec}(i)$ as the set of all successors of activity $i \in V$ in $G^{\prec}$, the path following rule (PF) with $\pi_i = l(i)$ and $l(i)$ as the maximal number of nodes on any longest directed path from node $i \in V$ to $n+1$ in project network N, and the maximal resource consumption rule (MRC) with $\pi_i = p_i \sum_{k \in \mathcal{R}_i} r_{ik}^d$. In contrast to the priority rules from the literature, the following rules make use of the properties of the partially renewable resources. For this, the maximal possible additional resource consumption $p_i\, r_{ik}^d$ by an activity in relation to the maximal remaining resource capacity $\bar{R}_k$ is considered with

$$\bar{R}_k := R_k - \sum_{i \in V_k} \underline{r}_{ik}^c(W, \mathrm{LS}_i^{\mathrm{UB}}(W))$$

and $\underline{r}_{ik}^c(W, d) := \min\{r_{ik}^c(\tau) : | : \tau \in W_i \cap [ES_i(W), d]\}$. We have tested the following two different versions. The total maximal additional relative resource consumption rule (TMAR) with

$$\pi_i = p_i \sum_{k \in \mathcal{R}_i : \bar{R}_k \neq 0} \frac{r_{ik}^d}{\bar{R}_k} + \sum_{k \in \mathcal{R}_i : \bar{R}_k = 0} 1,$$

and the average maximal additional relative resource consumption rule (AMAR) with $\pi_i = \pi'_i/|\mathcal{R}_i|$ and π'_i as the priority value of TMAR.

After the branching activity has been selected for some enumeration node, the last part of the search strategy is concerned with the generation and the ordering of the direct descendants. For the generation strategy, we distinguish between the possibilities either to generate all direct descendants (all) or to restrict the number of generated nodes by a maximal value (restr), where one and the same node must possibly be explored more than once. Furthermore, we have implemented different orders in which all start times in the reduced scheduling set T_i of branching activity $i \in \mathcal{E}$ are considered. The most intuitive alternative for this takes always the lowest start time from T_i that has not been used so far to generate a direct descendant node (LT). The other alternative assigns a priority value π_t to each start time $t \in T_i$ and considers all start times in T_i in an order of non-decreasing priority values (PV), where ties are broken on the basis of lower start times. In what follows, we present the best priority value we have found to order the start times in T_i. The corresponding priority value

$$\pi_t = \sum_{k \in \mathcal{R}_i} a_{ikt} + \frac{1}{4} \max_{k \in \mathcal{R}_i}(b_{ik})(t - ES_i(W))$$

with

$$a_{ikt} := \begin{cases} r^c_{ik}(t)/\bar{R}_k, & \text{if } \bar{R}_k \neq 0 \\ 1, & \text{otherwise} \end{cases}$$

and

$$b_{ik} := \begin{cases} r^d_{ik}/\bar{R}_k, & \text{if } \bar{R}_k \neq 0 \\ 1, & \text{otherwise} \end{cases}$$

is a combination of a priority value that is highly related to the TMAR rule and a penalty term that increases the priority value in a linear fashion based on the difference between start time t and the earliest W-feasible start time $ES_i(W)$. Finally, after all direct descendant nodes have been generated, the ordering strategy determines the order in which they are considered in the further course of the BnB. In this work, all generated descendant nodes are explored in an order of non-decreasing lower bounds on the project duration (LB), which has shown to provide good results in computational experiments.

In accordance with Watermeyer and Zimmermann (2020), we apply three different sets of consistency tests in our BnB. Set Γ^B contains the temporal- and resource-bound consistency test, Γ^D the temporal-bound and D-interval consistency test, and Γ^W the temporal and W-interval consistency test. $\gamma^\alpha_\beta(W)$ denotes the start time restriction that results if all consistency tests in Γ^β are applied on W for α iterations. $\alpha = \infty$ implies that the fixed point corresponding to Γ^β is determined. To be able to differentiate between the possibilities for the D-interval and W-interval consistency test to use all resources or only the resources that are demanded by activity i, we use the notations $\gamma^\alpha_\beta(W)[\mathcal{R}]$ and $\gamma^\alpha_\beta(W)[\mathcal{R}_i]$.

Algorithm 4: Branch-and-bound algorithm

Input: Instance of problem RCPSP/max-π
Output: Optimal schedule S^*

Determine distance matrix $D = (d_{ij})_{i,j \in V}$
$ES_i := d_{0i}$, $LS_i := -d_{i0}$ for all $i \in V$
$W_i := \{ES_i, \ldots, LS_i\}$ for all $i \in V$
Apply preprocessing on W
if $\mathcal{S}_T(W) = \emptyset$ **then** terminate ($\mathcal{S} = \emptyset$)
$LB^G := \min W_{n+1}$
$\mathcal{C} := \{0\}$ $S := \min \mathcal{S}_T(W)$ $LB := LB^G$
$\Omega := \{(\mathcal{C}, S, W, LB)\}$ $UB := \bar{d} + 1$
while $\Omega \neq \emptyset$ **do**
  Remove $(\mathcal{C}, S, W, LB)$ from stack Ω
  if $LB < UB$ **then**
    Apply consistency tests in Γ^D or Γ^W on W
    if $\mathcal{S}^{UB}_T(W) \neq \emptyset$ **then**
      Select activity $i \in \mathcal{E}$ and initialize $\Lambda := \emptyset$
      $\Theta_i := \{\tau \in W_i \mid r^c_k(S^{\mathcal{C}}) + r^c_{ik}(\tau) \leq R_k \text{ for all } k \in \mathcal{R}_i\}$
      Calculate T_i (Algorithm 2)
      while $T_i \neq \emptyset$ **do**
        Remove t from set T_i
        (according to the generation strategy)
        $S'_i := t$ $S' := \min \widetilde{\mathcal{S}}_T(W, i, S'_i)$
        $W' := (W_j \setminus [0, S'_j[)_{j \in V}$
        Apply consistency tests in Γ^B on W'
        if $\mathcal{S}^{UB}_T(W') \neq \emptyset$ **then**
          $S' := \min \mathcal{S}_T(W')$
          if $\exists j \in \mathcal{C} : S'_j > S_j$ **then**
            $\mathcal{C}' := \mathcal{C} \setminus \{j \in \mathcal{C} \mid S'_j > S_j\}$
          else if $S'_i = t$ **then**
            $\mathcal{C}' := \mathcal{C} \cup \{i\}$
          if $\mathcal{C}' = V$ **then**
            $S^* := S'$ $UB := S^*_{n+1}$
            $T_i := T_i \setminus [t+1, \infty[$
          else
            Compute lower bound LB'
            if $LB' < UB$ **then**
              $\Lambda := \Lambda \cup \{(\mathcal{C}', S', W', LB')\}$
      Put all nodes from Λ on stack Ω
      (according to the ordering strategy)
if $UB = \bar{d} + 1$ **then** terminate ($\mathcal{S} = \emptyset$)
else return S^*

In what follows, we outline the framework of our BnB, which is given in Algorithm 4. It should be noted that in order to simplify the presentation, we assume that a depth-first

search (DFS) is used and that all direct descendants of each enumeration node are generated at once (all). Accordingly, all other alternatives for the traversing and the generation strategy are omitted. In the first part of Algorithm 4, a preprocessing step is performed on the start time restriction W, for which we calculate the fixed point of set Γ^W considering all resources, meaning that $W{:=}\gamma_W^\infty(W)[\mathcal{R}]$ is set. In case that the preprocessing step cannot prove the infeasibility ($\mathcal{S}_T(W) \neq \emptyset$), the global lower bound LB^G is set to min W_{n+1}, the upper bound on the minimum project duration UB is set to $\bar{d}+1$, and the root node is initialized and put on stack Ω. In each iteration, an enumeration node $(\mathcal{C}, S, W, LB)$ is removed from stack Ω and it is checked if it could provide a solution with a better project duration than UB, i.e., $LB < UB$. In this case, consistency tests from set Γ^D or Γ^W are applied on the start time restriction W. If the consistency tests can show that the considered node and all its descendants cannot generate any feasible schedule with a better project duration than UB, i.e., $\mathcal{S}_T^{UB}(W){:=}\widehat{\mathcal{S}}_T(W, n+1, UB-1) = \emptyset$, the next enumeration node in Ω is considered. Otherwise, based on the branching strategy, the eligible set $\mathcal{E}$ is determined and the branching activity $i \in \mathcal{E}$ is selected, followed by the initialization of Λ, which is used to store all direct descendants of the considered enumeration node. In the next step, analogously to Algorithm 3, Θ_i and the reduced scheduling set T_i are calculated. The start times in T_i are considered in an order depending on the generation strategy. Given some start time t that has been removed from T_i, t is established as the earliest start time of the branching activity. It should be noted that line 19 of Algorithm 4 can be adapted as described in Sect. 4.3 to apply the pruning techniques UPT and ULT. After the initialization (and update) of start time restriction W', the consistency tests from set Γ^B are applied on W'. The direct descendant node corresponding to W' is directly pruned from the enumeration tree if $\mathcal{S}_T^{UB}(W') = \emptyset$. Otherwise, in case that the existence of any feasible schedule in $\mathcal{S}_T(W')$ with a better objective function value than UB cannot be excluded, it is checked if some activities have to be unscheduled or if the branching activity can be scheduled. If $\mathcal{C}' = V$ after the scheduling of the branching activity, a new best feasible solution $S^*{:=}S'$ has been found, UB is set to S^*_{n+1}, and all start times in T_i that are greater than t are removed, noticing that they cannot generate any better feasible solution. Otherwise, the lower bound LB' for the descendant node is calculated by $LB0^\pi$ or LBD^π. In case that $LB' < UB$, the node is added to the list Λ, which is used after the generation of all direct descendant nodes to put them on the stack Ω in an order of non-increasing values of their lower bounds LB'. The described procedure reiterates until there is no enumeration node left to be considered, i.e., $\Omega = \emptyset$. In case that $UB = \bar{d}+1$ at the end of the algorithm, we can state that there is no feasible solution, while otherwise, Algorithm 4 returns an optimal solution S^*.

6 Performance analysis

In this section, we evaluate the performance of our BnB. For this, we conduct computational experiments on different benchmark sets with partially renewable resources. To provide a comprehensive investigation, we compare our procedure with all available BnB from the literature for RCPSP/max-π and RCPSP/π. In a second step, we derive a priority rule-based approximation method from our new enumeration approach and evaluate its performance in comparison with the associated BnB.

The computational experiments have been conducted on a workstation with an Intel Core i7-8700 CPU with a clock pulse of 3.2 GHz and 64 GB RAM under Windows 10 on a single thread. The algorithms were all coded in C++ and compiled by the 64-bit Visual Studio 2017 C++-compiler.

6.1 Comparison of branch-and-bound algorithms

In the first part of the performance analysis, we compare our constructive BnB (CBB) with all available exact solution procedures from the literature for partially renewable resources. In the second part, we derive a schedule-generation scheme from our new enumeration approach and compare its results with those of CBB.

6.1.1 General temporal constraints

In this section, CBB is compared with the relaxation-based BnB (RBB) from Watermeyer and Zimmermann (2020) on instances with general temporal constraints. To the best of our knowledge, RBB represents the only BnB for RCPSP/max-π that is available in the open literature so far. For the comparison of CBB with RBB, we have conducted computational studies on a benchmark set that covers instances with $n = 10, 20, 50, 100, 200$ real activities, all of them with 30 partially renewable resources. The benchmark set UBOπ, which is available online,[1] is an adaptation of the well-known benchmark set UBO for RCPSP/max, which has been generated by the instance generator ProGen/max (Schwindt 1996, 1998). As described in Watermeyer and Zimmermann (2020), the test sets for RCPSP/max-π, denoted by UBO$n\pi$, were obtained by a replacement of the renewable resources by partially renewable resources that have been generated in accordance with the procedure in Schirmer (1999, Sect. 10).

Table 1 provides an overview of the settings we have used for CBB in the computational experiments depending on the instance size. The settings were determined by preliminary computational tests in the same way as for RBB in Watermeyer and Zimmermann (2020). Table 1 gives the set-

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

Table 1 Settings for the performance analysis

	UBO10π	UBO20π	UBO50π	UBO100π	UBO200π
Traversing strategy	DFS	SPS [2 s]	SPS [5 s]	SPS [5 s]	SPS [5 s]
Branching strategy	$\bar{C}$, MRC(max)	$\bar{C}$, MRC(max)	$\prec_{\mathrm{D}}$, STd$^{\mathrm{I}}$(min)	$\prec_{\mathrm{C}}$, PF(max)	$\prec_{\mathrm{C}}$, PF(max)
Generation strategy	restr-LT [10]	restr-LT [10]	restr-PV [5]	restr-PV [15]	restr-PV [15]
Ordering strategy	LB(min)	LB(min)	LB(min)	LB(min)	LB(min)
Consistency tests	γ_B^∞	γ_B^∞, $\gamma_D^1[\mathcal{R}_i, 2]$	γ_B^∞, $\gamma_W^1[\mathcal{R}_i]$	γ_B^∞, $\gamma_W^1[\mathcal{R}_i, 2]$	γ_B^∞
Lower bound	LBD$^\pi$	LBD$^\pi$	LBD$^\pi$	LB0$^\pi$	LB0$^\pi$
Pruning techniques	UPT	UPT	UPT+ULT	UPT+ULT	UPT+ULT

tings that have shown the best balance between the number of solved instances and instances whose solvability statuses were determined among all settings we have tested. It should be noted that we have also analyzed the performance of both BnB that were conducted with only one setting over all instances of benchmark set UBOπ. The corresponding results are given in the supplementary material of this paper (Online Resource 1). While the most terms in Table 1 are in line with Sect. 5, there are some additional specifications, which are explained in the following. The values in brackets in Table 1 give the time span for the scattered-path search, the maximal number of generated nodes in one branching step for the generation strategy, and the maximal search tree level on which the sets of consistency tests are applied. The values in parentheses indicate if lower (min) or greater (max) priority values or lower bounds are preferred. As already mentioned, Table 1 lists the settings that have shown the best balance between the number of instances that were solved and whose solvability status remained open among all settings we have tested. From Table 1 it follows that the restriction of the eligible set for the branching step in accordance with strict orders ($\prec_{\mathrm{D}}$, $\prec_{\mathrm{C}}$) is only beneficial for greater instances. Furthermore, the computational studies reveal that resource-based priority rules are preferable for small instances to select the branching activity, whereas temporal-based or network-based priority rules are better suited for greater instances. It is also worth mentioning that SPS and the usage of priority values for the generation of direct descendant nodes (PV) have both a great impact on the performance for greater instances. Finally, taking a look on the improving techniques, Table 1 shows that it is beneficial over all instances to use UPT and to calculate the fixed point γ_B^∞ in each enumeration node, while additional procedures can enhance the performance just for a few test sets.

Since for the greatest instances the improving techniques of CBB have shown to result in a significant increase in the number of instances whose solvability status could not be determined, we have implemented a warm-up phase (W). For this procedure, CBB is conducted with no improving technique for $n/10$ seconds, followed by CBB with the settings that are given in Table 1 (CBB+W).

Table 2 shows the performance of CBB and RBB. For the performance analysis, we have used a time limit of 300 seconds. The results for RBB are taken from Watermeyer and Zimmermann (2020), where RBB has been tested on the same workstation under the same conditions as CBB. In the third column, Table 2 gives the number of instances for which the earliest start time schedule *ES* is not optimal

Table 2 Performance of CBB and RBB (300 s)

		#nTriv	#opt	#feas	#inf	#unk	Δ^{lb} (%)	$\varnothing^{\mathrm{cpu}}$ (s)	$\varnothing^{\mathrm{cpu}}_{\mathrm{opt}}$ (s)	$\varnothing^{\mathrm{cpu}}_{\mathrm{inf}}$ (s)
UBO10π	CBB	693	534	534	159	0	53.43	0.065	0.067	0.056
	RBB	693	534	534	159	0	53.43	0.032	0.040	0.004
UBO20π	CBB	621	537	581	40	0	64.67	28.086	7.846	0.702
	RBB	621	500	578	40	3	65.09	46.149	8.076	8.006
UBO50π	CBB	527	183	491	5	31	88.16	198.016	13.774	26.827
	RBB	527	145	486	3	38	95.49	217.958	8.022	0.279
UBO100π	CBB	484	85	472	0	12	168.68	249.827	14.307	–
	RBB	484	79	465	0	19	174.30	254.409	20.681	–
UBO200π	CBB	466	95	449	0	17	222.93	243.429	22.502	–
	CBB+W	466	95	466	0	0	220.09	245.209	31.234	–
	RBB	466	79	466	0	0	224.03	253.934	28.271	–

(#nTriv), so-called non-trivial instances in line with Alvarez-Valdes et al. (2008). Since trivial instances can efficiently be solved to optimality, they are excluded from all investigations in the remainder of this work. The following columns list for each test set the number of instances for which an optimal solution is found and verified (#opt), infeasibility is shown (#inf), a feasible solution is found (#feas), or the solvability status remains unknown (#unk). The next columns give the average percentage deviation of the determined upper bound UB from the earliest time-feasible project termination ES_{n+1} (Δ^{lb}) and the average computing time ($\varnothing^{\mathrm{cpu}}$). The percentage deviation from ES_{n+1} is defined by zero for each instance that were shown to be infeasible. For comparison purposes both measures are given in relation to the number of all non-trivial instances. The last two columns provide the average computing time over all instances that were solved to optimality ($\varnothing^{\mathrm{cpu}}_{\mathrm{opt}}$) and have been shown to be infeasible ($\varnothing^{\mathrm{cpu}}_{\mathrm{inf}}$). Table 2 reveals a great dominance of CBB for test sets UBO20π, UBO50π, and UBO100π, whereas RBB is able to solve test set UBO10π in less computing time. The results for test set UBO200π show that CBB+W also clearly dominates RBB, while CBB is not capable to determine the solvability status for all instances without the warm-up phase. It should be noted that this result gives an important implication for constructive approximation methods for RCPSP/max-π that redundancies should be maintained in the solution procedure for greater instances. Since CBB+W was only able to improve the performance for UBO200π, the corresponding results for all other test sets are omitted. A closer look at the results shows that the intractability of the instances is strongly affected by a lower resource availability and a higher number of demanded resources per activity. These results are in line with general expectations, since the described characteristics increase the interdependence between the scheduling times of the activities. It should be noted that further performance tests with a time limit of 600 s showed that the gap between CBB and RBB over all test sets remained rather unchanged and that only a few more instances were solved by both procedures, respectively.

In order to evaluate the quality of the best found solutions by CBB for which the optimality could not be verified, Table 3 provides a comparison with the best solutions of RBB. For test set UBO200π the results of CBB+W are considered. The first part of Table 3 gives an overview about the number of instances for which a feasible solution has been found by at least one ($\#^{\cup}_{\mathrm{feas}}$) or by both procedures ($\#^{\cap}_{\mathrm{feas}}$), followed by the number of instances for which only CBB ($\#^{<}_{\mathrm{feas}}$) or RBB ($\#^{>}_{\mathrm{feas}}$) was able to find a feasible solution. For the test sets UBO20π, UBO50π, and UBO100π, CBB is able to find a feasible solution for more instances than RBB, where there is no instance for which only RBB detects a feasible solution. In the second part of Table 3, the quality of the feasible solutions is compared with each other. The first column gives the number of instances for which both procedures have found a feasible solution, but not both procedures could verify the optimality for ($\#^{\cap,\mathrm{nv}}_{\mathrm{feas}}$). These instances are subdivided into the number of instances with a better ($\#^{<}$), an equal ($\#^{=}$), or a worse ($\#^{>}$) found solution by CBB compared to RBB. The last two columns are concerned with the average deviations of the best found project durations by CBB from those of RBB, which are assumed to be given by S^{CBB}_{n+1} and S^{RBB}_{n+1}, respectively. In the first column, the average of the absolute deviation $S^{\mathrm{CBB}}_{n+1} - S^{\mathrm{RBB}}_{n+1}$ over all considered instances is given ($\Delta^{\mathrm{abs}}_{\mathrm{RBB}}$), while the second column depicts the average of the relative deviation $(S^{\mathrm{CBB}}_{n+1} - S^{\mathrm{RBB}}_{n+1})/S^{\mathrm{RBB}}_{n+1}$ from the best found project duration by RBB ($\Delta^{\mathrm{rel}}_{\mathrm{RBB}}$). The second part of Table 3 shows that CBB obtains better feasible solutions for more instances than RBB over all test sets. Furthermore, the last two columns indicate a dominance of CBB in terms of a better solution quality.

In order to illustrate the impact of the improving techniques on the performance of CBB, Table 4 shows the results for test set UBO20π with a time limit of 300 seconds if the search strategy in accordance with Table 1 is applied with different combinations of the given improving techniques. In the first two lines, the results of CBB are given if it is conducted without any improving technique, except that the lower bound $LB0^{\pi}$ is calculated in any enumeration node, termed basic version in the following. To investigate the benefit to calculate the reduced scheduling set T_i in each branching step, in Table 4 the results of two different basic versions of CBB are listed that consider the start times in Θ_i or T_i, respectively. In the following lines, the improving techniques from Table 1 are individually added to the basic version of CBB. The calculation of the reduced scheduling set as well as all improving techniques enhances the perfor-

Table 3 Comparison of the feasible solutions of CBB with RBB (300 s)

instance set	$\#^{\cup}_{\mathrm{feas}}$	$\#^{\cap}_{\mathrm{feas}}$	$\#^{<}_{\mathrm{feas}}$	$\#^{>}_{\mathrm{feas}}$	$\#^{\cap,\mathrm{nv}}_{\mathrm{feas}}$	$\#^{<}$	$\#^{=}$	$\#^{>}$	$\Delta^{\mathrm{abs}}_{\mathrm{RBB}}$	$\Delta^{\mathrm{rel}}_{\mathrm{RBB}}$ (%)
UBO10π	534	534	0	0	0	–	–	–	–	–
UBO20π	581	578	3	0	79	37	32	10	−2.37	−1.55
UBO50π	491	486	5	0	344	290	38	16	−16.87	−4.74
UBO100π	472	465	7	0	390	243	31	116	−14.51	−2.42
UBO200π	466	466	0	0	391	251	20	120	−20.27	−1.19

Table 4 Impact of components on the performance for test set UBO20π (300 s)

	#opt	#feas	#inf	#unk	Δ^{lb} (%)	$\varnothing^{\text{cpu}}$ (s)	$\varnothing^{\text{cpu}}_{\text{opt}}$ (s)	$\varnothing^{\text{cpu}}_{\text{inf}}$ (s)
BnB (basic version Θ_i)	120	546	3	72	81.20	246.446	29.134	49.019
BnB (basic version T_i)	142	560	5	56	79.28	235.378	25.654	65.402
+Preprocessing	214	567	22	32	75.22	193.991	23.165	0.493
+LBD$^{\pi}$	222	567	22	32	75.22	190.887	24.408	1.010
+Consistency tests	343	576	23	22	71.17	129.094	10.667	0.363
+UPT	537	581	40	0	64.67	28.086	7.846	0.702

mance of CBB with a significant reduction in the average deviation Δ^{lb} and computing time $\varnothing^{\text{cpu}}$.

6.1.2 Precedence constraints

In this section we investigate the performance of CBB on RCPSP/π benchmark sets. To evaluate the performance, CBB is compared with RBB and the only available BnB for RCPSP/π (BOT), which has been developed in Böttcher et al. (1999). Since the original code for BOT could not be provided to us, we have reimplemented BOT in line with Böttcher (1995) and Böttcher et al. (1999). As preliminary tests have shown, the best results for BOT are obtained if the feasibility bounds FB1 and FB2 as described in Böttcher et al. (1999) are used. Hence, we have applied both feasibility bounds in all computational experiments on BOT.

The first benchmark set contains 2160 instances with 10 real activities (P10π) and 250 instances with 15, 20, 25, and 30 real activities (P15π, P20π, P25π, P30π), respectively. All of them were generated with 30 partially renewable resources. These test sets have been used in Alvarez-Valdes et al. (2006, 2008) for a performance analysis and were provided to them by the authors of Böttcher et al. (1999). Table 5 shows the results of an experimental performance analysis on the Böttcher benchmark set with a time limit of 300 seconds. For CBB and RBB we have used the settings of test set UBO20π, except for test set P25π, for which we have conducted the computational tests on CBB with the settings of test set UBO50π. Table 5 shows that both CBB and RBB dominate BOT over all instances, while only small differences are given between CBB and RBB, except that RBB tends to show lower computing times.

The second RCPSP/π benchmark set was generated by Schirmer (1999) and was later extended by Alvarez-Valdes et al. (2006, 2008). The test sets of Schirmer (1999) cover 960 instances with 10, 20, 30, and 40 real activities (J10π, J20π, J30π, J40π), respectively, with 30 partially renewable resources. Later, Alvarez-Valdes et al. (2006, 2008) added a

Table 5 Performance on the Böttcher benchmark set (300 s)

		#nTriv	#opt	#feas	#inf	#unk	Δ^{lb} (%)	$\varnothing^{\text{cpu}}$ (s)	$\varnothing^{\text{cpu}}_{\text{opt}}$ (s)	$\varnothing^{\text{cpu}}_{\text{inf}}$ (s)
	CBB	2108	827	827	1281	0	11.20	0.055	0.056	0.054
P10π	RBB	2108	827	827	1281	0	11.20	0.007	0.007	0.007
	BOT	2108	827	827	1281	0	11.20	0.023	0.023	0.023
	CBB	204	188	188	16	0	37.23	1.704	1.845	0.051
P15π	RBB	204	188	188	16	0	37.23	1.948	2.114	0.002
	BOT	204	181	181	16	7	38.19	12.717	2.727	0.036
	CBB	165	139	142	17	6	52.69	16.464	0.112	0.052
P20π	RBB	165	139	142	17	6	52.66	16.920	0.660	0.002
	BOT	165	136	139	16	10	54.71	27.124	3.974	2.187
	CBB	136	112	116	14	6	69.31	22.155	0.109	0.062
P25π	RBB	136	112	115	14	7	68.90	22.074	0.018	0.010
	BOT	136	105	111	11	14	74.32	46.535	0.465	25.457
	CBB	122	104	104	8	10	91.99	24.655	0.072	0.053
P30π	RBB	122	104	104	8	10	91.99	24.615	0.029	0.003
	BOT	122	98	104	3	15	98.53	53.327	2.095	0.196

Table 6 Performance on the SAV benchmark set (300 s)

		#nTriv	#opt	#feas	#inf	#unk	Δ^{lb} (%)	$\varnothing^{\text{cpu}}$ (s)	$\varnothing^{\text{cpu}}_{\text{opt}}$ (s)	$\varnothing^{\text{cpu}}_{\text{inf}}$ (s)
	CBB	808	803	803	5	0	10.37	0.063	0.063	0.055
J10π	RBB	808	803	803	5	0	10.37	0.060	0.060	0.052
	BOT	808	802	802	5	1	10.54	0.541	0.171	0.041
	CBB	565	563	565	0	0	5.22	1.965	0.906	–
J20π	RBB	565	564	565	0	0	5.22	2.313	1.785	–
	BOT	565	509	561	0	4	7.93	35.533	6.436	–
	CBB	453	431	453	0	0	4.37	17.603	3.189	–
J30π	RBB	453	427	453	0	0	4.18	20.723	3.717	–
	BOT	453	345	435	0	18	14.18	75.767	5.573	–
	CBB	386	347	386	0	0	6.06	35.153	5.386	–
J40π	RBB	386	341	386	0	0	6.07	38.599	4.103	–
	BOT	386	261	363	0	23	21.24	100.109	4.376	–
	CBB	346	269	346	0	0	8.91	73.353	8.476	–
J60π	RBB	346	268	346	0	0	14.06	69.313	2.172	–
	BOT	346	186	309	0	37	40.50	140.073	2.502	–

test set with 960 instances, each of them with 60 real activities (J60π) and 30 partially renewable resources. It should be noted that 9 instances of test set J10π, which have been proven to be infeasible in Schirmer (1999, Sect. 10.4), could not be provided to us, so that they are not part of the performance analysis. In the following, we summarize all instances from Schirmer (1999) and Alvarez-Valdes et al. (2006, 2008) under the term SAV benchmark set. In Table 6, the results of the computational tests on the SAV benchmark set with a time limit of 300 seconds are given. As for the Böttcher instances, we have used the settings of UBO20π for CBB and RBB for the computational experiments, with the only exception that CBB has been conducted with the settings of UBO50π for test set J60π. Table 6 reveals that both CBB and RBB outperform BOT on the SAV benchmark set. Furthermore, Table 6 shows slightly better results for CBB compared to those of RBB.

6.2 Schedule-generation scheme

The enumeration approach of CBB, which is based on a serial schedule-generation scheme (SGS), can directly be used as a framework for a priority rule-based approximation method. For this, in each branching step only one of the direct descendant nodes has to be chosen in accordance with a priority rule. In what follows, we investigate the performance of a regret-based biased random sampling method that makes use of a regret measure that has been introduced by Drexl (1991) and Drexl and Grünewald (1993).

Considering the extended enumeration scheme in Algorithm 3, each activity $i \in \mathcal{E}$ and each start time $t \in T_i$ that could be selected in a branching step is assigned a priority value π_i (π_t). [2] For simplicity, let $\mathcal{D}$ be the set of all candidates for each selection and s some candidate (activity i or start time t). Then, in accordance with Kolisch (1996), each candidate s is assigned a regret value

$$\rho_s := \begin{cases} \max\limits_{h \in \mathcal{D}} \pi_h - \pi_s, & \text{if ext} = \min \\ \pi_s - \min\limits_{h \in \mathcal{D}} \pi_h, & \text{if ext} = \max \end{cases}$$

depending on whether lower (ext = min) or greater (ext = max) priority values are preferred and a selection probability

$$\psi_s := \frac{(\rho_s + 1)^\alpha}{\sum\limits_{h \in \mathcal{D}} (\rho_h + 1)^\alpha}.$$

The eligible set $\mathcal{E}$ and the priority values for the start times are determined in accordance with the settings for CBB. Based on preliminary tests, parameter α for the calculation of the selection probability ψ_s is set to $\alpha = 2$ for $n = 10$, $\alpha = 4$ for $10 < n \leq 30$, $\alpha = 16$ for $30 < n \leq 60$, and $\alpha = 32$ for all greater instances ($n > 60$).

Table 7 shows the performance of SGS on test sets UBO10π, UBO20π, and UBO50π for $Z = 100$ and $Z =$

[2] In contrast to Algorithm 3, the eligible set $\mathcal{E}$ is considered for the activity selection in order to provide the possibility to use the distance order or cycle order as described in Sect. 5.

Table 7 Performance of SGS on test sets UBO10π, UBO20π, and UBO50π

			$Z = 100$				$Z = 1000$			
		#inst	#nfeas	Δ^{lb} (%)	Δ^{CBB} (%)	$\varnothing^{\text{cpu}}$ (s)	#nfeas	Δ^{lb} (%)	Δ^{CBB} (%)	$\varnothing^{\text{cpu}}$ (s)
	SGS(Θ_i)	534	17	81.82	9.46	0.091	8	75.44	4.27	0.098
	SGS(T_i)	534	12	79.01	7.50	0.093	5	74.88	4.21	0.202
UBO10π	+RB	534	4	75.13	4.55	0.088	1	71.92	1.87	0.194
	+UPT	534	0	70.90	1.04	0.078	0	69.66	0.18	0.131
	CBB	534	0	69.34	0.00	0.067				
	SGS(Θ_i)	581	34	93.28	21.59	0.090	22	82.34	10.80	0.493
	SGS(T_i)	581	27	88.21	16.56	0.109	18	79.59	8.32	0.735
UBO20π	+RB	581	16	82.14	11.12	0.182	12	75.52	4.90	0.709
	+UPT	581	9	74.75	3.92	0.119	5	72.18	2.08	0.371
	CBB	581	0	69.12	0.00	29.972				
	SGS(Θ_i)	522	34	147.76	54.17	1.078	32	109.82	18.95	6.663
	SGS(T_i)	522	32	124.20	33.38	1.363	32	107.14	16.41	8.621
UBO50π	+RB	522	31	123.73	32.93	1.341	31	106.80	16.11	8.800
	+UPT	522	74	106.46	13.87	0.536	54	97.89	6.27	3.940
	CBB	522	31	89.00	0.00	199.656				

Table 8 Performance of SGS on test sets UBO100π and UBO200π

			$Z = 10$				$Z = 100$			
		#inst	#nfeas	Δ^{lb} (%)	Δ^{CBB} (%)	$\varnothing^{\text{cpu}}$ (s)	#nfeas	Δ^{lb} (%)	Δ^{CBB} (%)	$\varnothing^{\text{cpu}}$ (s)
	SGS(Θ_i)	484	18	350.92	120.75	1.243	18	315.76	103.12	10.575
	SGS(T_i)	484	18	259.00	71.99	1.519	18	227.62	53.56	13.567
UBO100π	+RB	484	18	259.06	72.03	1.533	17	226.96	52.85	13.627
	+UPT	484	224	261.55	45.04	0.503	130	217.47	18.39	2.794
	CBB	484	12	168.68	0.00	249.827				
	SGS(Θ_i)	466	0	473.45	152.90	7.964	0	452.01	143.49	70.409
	SGS(T_i)	466	0	335.24	85.71	8.260	0	311.78	75.28	76.583
UBO200π	+RB	466	0	335.28	85.70	8.315	0	311.76	75.26	76.996
	+UPT	466	189	340.11	65.57	3.327	76	274.49	30.47	22.722
	CBB+W	466	0	220.09	0.00	245.209				

1000 iterations. In order to evaluate the impact of the reduced scheduling set T_i, the results for SGS are given for the case that Θ_i or T_i is considered for the start time selection, denoted by SGS(Θ_i) and SGS(T_i). Additionally, the results for SGS(T_i) are provided if the resource-bound (RB) consistency test (+RB) or the RB consistency test with the pruning technique UPT (+UPT) is applied. For comparison purposes, in the last row for each test set the results for CBB are given. The computational studies were conducted on all non-trivial instances that could not be shown to be infeasible by CBB (#inst). Δ^{lb} and $\varnothing^{\text{cpu}}$ are given in relation to these instances, while Δ^{CBB} represents the average percentage deviation of the best found solution by SGS from the shortest project duration that has been determined by CBB in relation to the number of instances for which SGS was able to find a feasible solution. For this, it should be noted that all instances, for which SGS was able to determine a feasible solution, were feasibly solved by CBB as well. Finally, the last measure is given by the number of instances for which SGS was not able to find a feasible solution (#nfeas).

Table 7 shows the best results for SGS that were obtained among all priority rules for the activity selection that are described in Sect. 5. The corresponding priority rules that have been applied on test sets UBO10π, UBO20π, and UBO50π are, respectively, given by LST, LSTd, and STd$^{\text{I}}$ with ext = min. It can be seen from Table 7 that the calculation of the reduced scheduling set T_i always results in a better performance for SGS and that the RB consistency

test is able to improve the solution procedure, but with a significant greater impact on test sets UBO10π and UBO20π. This observation might be explained by the reduction of the start times that are not part of any feasible solution by the RB consistency test whose effectiveness seems to increase if less possible start times are considered. In contrast, the additional application of the pruning technique UPT could only obtain better results for small instances, while the number of instances for which no feasible solution was determined is getting greater for test set UBO50π. The negative impact of UPT might be assumed to be caused by the reduction of redundancies that tend to increase the probability for unfavorable scheduling decisions if the number of activities is getting greater. The comparison with CBB reveals that SGS performs only reasonable on small instances (UBO10π, UBO20π) if the improving techniques are applied, while the best solutions for test set UBO50π (+RB) deviate more than 16% on average from the solutions of CBB even if $Z = 1000$ iterations are conducted.

Due to the significant increase in the computing time, Table 8 provides the results of SGS on test sets UBO100π and UBO200π for $Z = 10$ and $Z = 100$ iterations. The priority rules ST and STd with ext $=$ min were applied on the test sets UBO100π and UBO200π, respectively. In accordance with the results for test set UBO50π, the application of UPT leads to an increase in the number of instances for which SGS is not able to determine a feasible solution. Furthermore, the RB consistency test can at most slightly improve the performance, while the advantage of the reduced scheduling set T_i even tends to increase for greater instances. The best solutions of SGS clearly deviate from the shortest project durations determined by CBB on average, so that SGS seems to be not well suited to solve large instances.

The results of SGS on the Böttcher and SAV benchmark set show similar trends as for the test sets UBO10π and UBO20π. For details, we refer the reader to the supplementary material of this paper.

In conclusion, SGS shows a reasonable performance on small instances if the RB consistency test and the pruning technique UPT are applied, while the procedures lack to improve the performance for greater instances, which might be assumed to be caused by the increase in the number of possible start times in each scheduling step. In our opinion, a promising way to improve the performance of SGS on greater instances might be to implement procedures that store information about the decisions that were made in previous iterations. One way for this might be given by adapting the construction procedure of the search tree that has been presented for CBB.

7 Conclusions

We have presented a branch-and-bound algorithm (BnB) for the resource-constrained project duration problem with partially renewable resources and general temporal constraints (RCPSP/max-π) that is based on a serial schedule-generation scheme. For the first time it has been shown that it is sufficient to consider only a subset of all resource-feasible start times in each branching step. By an extension of the enumeration scheme by start time domains, improving techniques from the literature could be included. Furthermore, we were able to devise new pruning techniques to prevent redundancies with a significant impact on the performance.

In a comprehensive experimental performance analysis we have compared our exact solution procedure with all BnB that are available in the open literature for partially renewable resources. The computational experiments could reveal a great dominance of our BnB for RCPSP/max-π. The favorable performance could also be confirmed for instances that are restricted to precedence constraints with significant better results compared to the only available BnB for RCPSP/π. In a second step, we investigated a directly derived schedule-generation scheme from our new enumeration approach. It has been shown that the approximation procedure obtains reasonable results for small instances, while a limitation for large instances became apparent.

As the computational experiments have shown, the performance of a BnB for RCPSP/max-π is strongly influenced by the way to enumerate the candidate solutions. Therefore, the investigation of further enumeration schemes seems to be a promising field for future research. Moreover, the experiments could demonstrate that there is a great need for new types of approximation methods that goes beyond classical priority rule-based generation schemes.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s10951-022-00735-9.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

Ahuja, R. K., Magnanti, T. L., & Orlin, J. B. (1993). *Network flows: Theory, algorithms, and applications*. Prentice-Hall.

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

Alvarez-Valdes, R., Crespo, E., Tamarit, J. M., & Villa, F. (2008). GRASP and path relinking for project scheduling under partially renewable resources. *European Journal of Operational Research, 189*(3), 1153–1170.

Alvarez-Valdes, R., Tamarit, J. M., & Villa, F. (2015). Partially renewable resources. In C. Schwindt & J. Zimmermann (Eds.), *Handbook on project management and scheduling* (Vol. 1, pp. 203–227). Springer.

Böttcher, J. (1995). Projektplanung (project scheduling): ein exakter Algorithmus zur Lösung des Problems mit partiell erneuerbaren Ressourcen (an exact algorithm for the problem with partially renewable resources). Diploma thesis, University of Kiel.

Böttcher, J., Drexl, A., Kolisch, R., & Salewski, F. (1999). Project scheduling under partially renewable resource constraints. *Management Science, 45*(4), 543–559.

Brucker, P., & Knust, S. (2003). Lower bounds for resource-constrained project scheduling problems. *European Journal of Operational Research, 149*(2), 302–313.

Demeulemeester, E., & Herroelen, W. (1992). A branch-and-bound procedure for the multiple resource-constrained project scheduling problem. *Management Science, 38*(12), 1803–1818.

Dorndorf, U., Pesch, E., & Phan-Huy, T. (2000a). Constraint propagation techniques for the disjunctive scheduling problem. *Artificial Intelligence, 122*(1), 189–240.

Dorndorf, U., Pesch, E., & Phan-Huy, T. (2000b). A time-oriented branch-and-bound algorithm for resource-constrained project scheduling with generalised precedence constraints. *Management Science, 46*(10), 1365–1384.

Drexl, A. (1991). Scheduling of project networks by job assignment. *Management Science, 37*(12), 1590-1602.

Drexl, A., & Grünewald, J. (1993). Nonpreemptive multi-mode resource-constrained project scheduling. *IIE Transactions, 25*(5), 74–81.

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

Kolisch, R. (1996). Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. *European Journal of Operational Research, 90*(2), 320–333.

Neumann, K., Nübel, H., & Schwindt, C. (2000). Active and stable project scheduling. *Mathematical Methods of Operations Research, 52*(3), 441–465.

Neumann, K., Schwindt, C., & Zimmermann, J. (2003). *Project scheduling with time windows and scarce resources* (2nd ed.). Springer.

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

Schwindt, C. (1996). Generation of resource-constrained project scheduling problems with minimal and maximal time lags. Report WIOR-489, University of Karlsruhe.

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

Stinson, J. P., Davis, E. W., & Khumawala, B. M. (1978). Multiple resource-constrained scheduling using branch and bound. *AIIE Transactions, 10*(3), 252–259.

Talbot, F. B., & Patterson, J. H. (1978). An efficient integer programming algorithm with network cuts for solving resource-constrained scheduling problems. *Management Science, 24*(11), 1163–1174.

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