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

A preprint of the paper
https://link.springer.com/article/10.1007/s10951-022-00735-9
doi: 10.1007/s10951-022-00735-9

Kai Watermeyer
Operations Research Group
Clausthal University of Technology
Julius-Albert-Str. 2
38678 Clausthal-Zellerfeld, Germany

Jürgen Zimmermann
Operations Research Group
Clausthal University of Technology
Julius-Albert-Str. 2
38678 Clausthal-Zellerfeld, Germany

May 22, 2020

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 the activities, even more aspects of real-life projects can be taken into account. We present a branch-and-bound algorithm for the stated problem which is based on a serial schedule-generation scheme. Besides some consistency tests and lower bounds, which are integrated in the solution process to improve the performance, we have also developed techniques which are able to prevent redundancies in the course of the enumeration. In a comprehensive experimental performance analysis we compare our exact solution procedure with all available branch-and-bound algorithms from the literature for partially renewable resources on benchmark test sets. The results of the computational study demonstrate the efficiency of our branch-and-bound algorithm.

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 which 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 a project scheduling problem by Böttcher et al. (1999). The corresponding problem, which is denoted by RCPSP/π, is a generalization of the classical resource-constrained project scheduling problem (RCPSP). The motivation for the extension of the 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 dissolved by partially renewable resources by assigning the availability of a resource to multiple subsets of time periods which can be seen to integrate the decision about the availability of a resource for a specific time period in the scheduling process. Examples for the application of the RCPSP/π can be found in Böttcher et al. (1999) for the flexible planning of lunch breaks in a company, 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 the RCPSP/π. In Böttcher et al. (1999) and Schirmer (1999), priority rule methods for the RCPSP/π are investigated. The works of Alvarez-Valdes et al. (2006, 2008, 2015) are devoted to a GRASP and a scatter search algorithm, and in Schirmer (1999) different local-search procedures are considered. To the best of our knowledge, the only exact solution procedure for the RCPSP/π is given in Böttcher et al. (1999), based on a branch-and-bound approach developed by Talbot and Patterson (1978).

For the first time, Watermeyer and Zimmermann (2020) have extended the RCPSP/π by taking minimum and maximum time lags between the start times of activities into account in order to cover more aspects of real-life projects. Watermeyer and Zimmermann (2020) provide a branch-and-bound approach for the problem, denoted by RCPSP/max-π, which is based on the solution of a resource relaxation in each enumeration node. In this work we present a branch-and-bound algorithm for the RCPSP/max-π which is based on an alternative enumeration approach which schedules all activities of the project successively.

The remainder of this paper is organized as follows. Section 2 provides a formal description of the RCPSP/max-π. In Sect. 3 we discuss the enumeration scheme of our branch-and-bound algorithm, where in Sect. 4 two different implementations for the branching step are presented. In Sect. 5, improving techniques are considered, followed by Sect. 6 which describes the branch-and-bound procedure. In Sect. 7 we present the results of a comprehensive experimental performance analysis and provide some conclusions in Sect. 8.

2 Problem description

The 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 $r^d_{ik} \in \mathbb{Z}_{\geq 0}$ 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 $S_{n+1} \leq \bar{d}$, where the set of all time-feasible schedules is denoted by $\mathcal{S}_T$. The resource constraints of the RCPSP/max-π are given by the resource capacities $R_k \in \mathbb{Z}_{\geq 0}$ of all partially renewable resources $k \in \mathcal{R}$, where 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 $i \in V_k := \{i \in V \mid r^d_{ik} > 0\}$ over the time periods in Π_k have 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 $r^u_{ik}(S_i) := |]S_i, S_i + p_i] \cap \Pi_k|$. Based on the resource usage, the resource consumption of a resource $k \in \mathcal{R}$ by an activity $i \in V$ follows directly with $r^c_{ik}(S_i) := r^u_{ik}(S_i) \cdot r^d_{ik}$, so that the resource constraints can be stated by $\sum_{i \in V} r^c_{ik}(S_i) \leq R_k$ for all $k \in \mathcal{R}$. In the following we call a schedule S which 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 the RCPSP/max-π is to determine a feasible schedule S^* with the lowest project duration among all feasible schedules $S \in \mathcal{S}$ which can be 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 which assigns the project duration to each feasible schedule S. In the following, we call a feasible schedule S which 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 which turned out to be more appropriate for theoretical issues.

3 Enumeration scheme

In general, the enumeration scheme of a branch-and-bound algorithm 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 branch-and-bound procedure 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), which provides a serial schedule-generation scheme (SGS) for the RCPSP/max, i.e., the RCPSP with general temporal constraints.

The construction procedure of the directed outtree corresponding to the enumeration scheme of our branch-and-bound algorithm 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 which 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 so-called 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 branch-and-bound procedure. 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 which have still to be explored and Φ to gather all feasible schedules which have been generated 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-$\pi$
Output: Set $\Phi$ of candidate schedules

1  Determine distance matrix $D = (d_{ij})_{i,j\in V}$
2  Set $ES_i := d_{0i}$, $LS_i := -d_{i0}$ for all $i \in V$
3  $\mathcal{C} := \{0\}$    $S := ES$
4  $\Omega := \{(\mathcal{C}, S)\}$    $\Phi := \emptyset$
5  while $\Omega \neq \emptyset$ do
6      Remove $(\mathcal{C}, S)$ from set $\Omega$
7      if $\mathcal{C} = V$ then
8          $\Phi := \Phi \cup \{S\}$
9      else
10         Select activity $i \in \bar{\mathcal{C}}$
11         $\Theta_i := \{\tau \in \{S_i, \ldots, LS_i\} \mid r^c_k(S^{\mathcal{C}}) + r^c_{ik}(\tau) \leq R_k$ for all $k \in \mathcal{R}_i\}$
12         Compute $T_i := ReducedSchedulingSet(\Theta_i)$
13         forall $t \in T_i$ do
14             $S'_i := t$    $S' := (\max(S_j, S'_i + d_{ij}))_{j\in V}$
15             if $\exists j \in \mathcal{C} : S'_j > S_j$ then
16                 $\mathcal{C}' := \mathcal{C} \setminus \{j \in \mathcal{C} \mid S'_j > S_j\}$
17             else
18                 $\mathcal{C}' := \mathcal{C} \cup \{i\}$
19             $\Omega := \Omega \cup \{(\mathcal{C}', S')\}$
20 return $\Phi$
```

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, where 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. In Schirmer (1999, Theorem 9.5) it has already been shown for the RCPSP/π that in general, the step-wise scheduling of all activities of the project at their earliest feasible start times does not guarantee to obtain an optimal schedule. As a consequence, all SGS for the RCPSP/π in the literature select the start time t for each not currently scheduled activity $i \in \bar{\mathcal{C}}$ out of the set of all feasible start times of activity i (which could not be eliminated by consistency tests) (see Schirmer, 1999; Alvarez-Valdes et al., 2006, 2008, 2015). In what follows, we show that it is sufficient to consider only a subset of Θ_i for some not currently scheduled activity $i \in \bar{\mathcal{C}}$, so that the generation of at least one optimal schedule is still guaranteed. As it is shown later on, due to the maximum time lags between the activities of the project, the restriction of set Θ_i requires the implementation of an unscheduling step. In the following, we call the respective subset the reduced scheduling set of Θ_i which is defined by $T_i := \{\tau \in \Theta_i \mid \nexists \tau' \in [0, \tau[\cap \Theta_i : r^u_{ik}(\tau) \geq r^u_{ik}(\tau') \text{ for all } k \in \mathcal{R}_i\}$. Roughly speaking, by using the reduced scheduling set T_i, a delay of the start time of activity $i \in \bar{\mathcal{C}}$ is only accepted if it results in a lower resource consumption for at least one resource $k \in \mathcal{R}_i$ with respect to all lower start times in Θ_i. For the moment, we assume that the reduced scheduling set T_i is given. Later on, we discuss two different algorithms to calculate T_i in Sect. 4. 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 the update of the earliest time-feasible start times for all activities of the project if it is assumed that activity i starts at time t which is given by $S' := (\max(S_j, S'_i + d_{ij}))_{j \in V}$. Since accordingly, for each start time $t \in T_i$ the minimum time lag to all currently scheduled activities is satisfied, i.e., $S_j + d_{ji} \leq t$ for all $j \in \mathcal{C}$, in case that start time 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 $S'_j > S_j$ has to be unscheduled which is obtained by $\mathcal{C}' := \mathcal{C} \setminus \{j \in \mathcal{C} \mid S'_j > S_j\}$. 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, which means that $t \leq \min_{j \in \mathcal{C}} LS_i(S_j)$, activity i is scheduled at start time $S'_i = t$ which is established by $\mathcal{C}' := \mathcal{C} \cup \{i\}$. 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. Accordingly, 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.

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 $\mathcal{AS}$, where 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 $i \in V$. Since obviously there is at least one optimal schedule which is active for each instance with $\mathcal{S} \neq \emptyset$, Theorem 1 implicates the completeness of Algorithm 1, i.e., $\mathcal{S} \neq \emptyset \Leftrightarrow \Phi \cap \mathcal{OS} \neq \emptyset$. Finally, Lemma 3 establishes that the enumeration scheme terminates after a finite number of iterations.

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

Proof. It is easy to verify that 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 1 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. □

Lemma 1. *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 $(\mathcal{C}', S')$ which 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, $S_i^f \in \Theta_i$ can easily be derived from $S_i \leq S_i^f \leq LS_i$ and $r_k^c(S^{\mathcal{C}}) + r_{ik}^c(S_i^f) \leq r_k^c(S^f) \leq R_k$ for all $k \in \mathcal{R}_i$. Since $S_i^f \in \Theta_i$, from Lemma 2 we get $t := \min\{\tau \in \Theta_i \mid r_{ik}^u(\tau) \leq r_{ik}^u(S_i^f)$ for all $k \in \mathcal{R}_i\} \in T_i$, so that $t \leq S_i^f$ and $r_{ik}^u(t) \leq r_{ik}^u(S_i^f)$ 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 $t \leq S_i^f$ $(t + d_{ij} \leq S_i^f + d_{ij} \leq S_j^f$ for all $j \in V)$ and the conditions $r_{jk}^u(S_j') \leq r_{jk}^u(S_j^f)$ 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. □

Lemma 2. *Let T_i be the reduced scheduling set of $\Theta_i \subseteq \{0, 1, \ldots, \bar{d}\}$. Then T_i contains exactly all lowest scheduling times $t \in \Theta_i$ which satisfy $r_{ik}^u(t) \leq r_{ik}^u(\tau)$ for any $\tau \in \Theta_i$ and all $k \in \mathcal{R}_i$, i.e., $T_i = T_i^{\cup} := \bigcup_{\tau \in \Theta_i}\{\min\{\tau' \in \Theta_i \mid r_{ik}^u(\tau') \leq r_{ik}^u(\tau)$ 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 $t = \min\{\tau \in \Theta_i \mid r_{ik}^u(\tau) \leq r_{ik}^u(t)$ for all $k \in \mathcal{R}_i\}$, so that $T_i \subseteq T_i^{\cup}$ is given. Next, let $\tau \in \Theta_i$ and $t := \min\{\tau' \in \Theta_i \mid r_{ik}^u(\tau') \leq r_{ik}^u(\tau)$ for all $k \in \mathcal{R}_i\}$ be given and assume $t \notin T_i$. Since $t \notin T_i$ implies $t > \min\{\tau' \in \Theta_i \mid r_{ik}^u(\tau') \leq r_{ik}^u(\tau)$ 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 3. *Algorithm 1 generates at most $\bar{d}^{\,\bar{d}|V|}$ 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 trivially bounded from above by the maximum project duration $\bar{d}$, an upper bound for the maximum depth of the enumeration tree is given by $\bar{d}|V|$. Accordingly, an upper bound for the maximum number of generated nodes is given by $\bar{d}^{\,\bar{d}|V|}$ taking into consideration that the number of start times in T_i is bounded from above by $\bar{d}$. □

4 Reduced scheduling set

This section is concerned with the calculation of the reduced scheduling set T_i of set Θ_i which contains the resource-feasible start times of activity $i \in \bar{\mathcal{C}}$ which is chosen for the augmentation of a partial schedule $S^{\mathcal{C}}$ in Algorithm 1. In what follows, we describe two different procedures to calculate $T_i := \{\tau \in \Theta_i \mid \nexists \tau' \in [0, \tau[\cap \Theta_i : r_{ik}^u(\tau) \geq r_{ik}^u(\tau')$ for all $k \in \mathcal{R}_i\}$.

Algorithm 2: Reduced scheduling set (v1)

```
Input: Scheduling set $\Theta_i$
Output: Reduced scheduling set $T_i$

1  $T_i := \emptyset$   $t := \min \Theta_i$   $element := true$
2  while $t < \infty$ do
3      forall $\tau \in T_i$ do
4          $element := false$
5          forall $k \in \mathcal{R}_i$ do
6              if $r_{ik}^u(t) < r_{ik}^u(\tau)$ then
7                  $element := true$
8                  break
9          if $element = false$ then
10             break
11     if $element = true$ then
12         $T_i := T_i \cup \{t\}$
13     $t := \min\{\tau \in \Theta_i \mid \tau > t\}$
14 return $T_i$
```

The first procedure is sketched in Algorithm 2, where $\min \emptyset := \infty$ is defined to simplify the representation. In the course of the procedure, variable t serves as the start time from Θ_i which is considered in the current iteration while the 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, which means 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 easily be verified by noticing that for each start time $\tau \in \Theta_i$, which 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 $t \in \Theta_i$ which is not added to T_i, 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 or rather returns the reduced scheduling set of Θ_i.

In contrast to Algorithm 2, which in the worst case compares the resource usage of each start time $t \in \Theta_i$ with the resource usage of all lower start times $\tau \in T_i$, the second algorithm to calculate T_i makes use of insights concerned with the course of the resource usage of an activity over its start times. In what follows, we show that the course of the resource usage of any resource $k \in \mathcal{R}_i$ by an activity $i \in V$ is constant within each of a number of time intervals covering the whole planning horizon $H := \{0, 1, \ldots, \bar{d}\}$, whose number is bounded by a polynomial function in the instance length. As a consequence, it is sufficient to store the resource usage $r^u_{ik}(\tau)$ of a resource $k \in \mathcal{R}_i$ by an activity $i \in V$ for a polynomially bounded number of times $\tau \in \Psi \subseteq H$ to be able to calculate the resource usage for any start time $t \in H$. In the following we call $I := \{a, a+1, \ldots, b\} \subseteq \Pi_k$ with $a-1, b+1 \notin \Pi_k$ a component of Π_k and denote by $\mathcal{U}^s_k := \{\sigma \mid \sigma \notin \Pi_k \wedge \sigma + 1 \in \Pi_k\}$ ($\mathcal{U}^e_k := \{\sigma \mid \sigma \in \Pi_k \wedge \sigma + 1 \notin \Pi_k\}$) the set of the start (end) times of all components in Π_k. Furthermore, we call $\Delta^u_{ik}(\tau) := r^u_{ik}(\tau+1) - r^u_{ik}(\tau)$ the resource usage change of resource $k \in \mathcal{R}_i$ by activity $i \in V$ at start time τ. Then in line with Watermeyer and Zimmermann (2020), the relations

$$\begin{aligned}
\tau + 1 \in \Pi_k \wedge \tau + p_i + 1 \in \Pi_k \quad &\Rightarrow \quad \Delta^u_{ik}(\tau) = \ 0 \\
\tau + 1 \in \Pi_k \wedge \tau + p_i + 1 \notin \Pi_k \quad &\Rightarrow \quad \Delta^u_{ik}(\tau) = \text{-}1 \\
\tau + 1 \notin \Pi_k \wedge \tau + p_i + 1 \in \Pi_k \quad &\Rightarrow \quad \Delta^u_{ik}(\tau) = \ 1 \\
\tau + 1 \notin \Pi_k \wedge \tau + p_i + 1 \notin \Pi_k \quad &\Rightarrow \quad \Delta^u_{ik}(\tau) = \ 0
\end{aligned}$$

for each start time $\tau \in H$ imply that it is sufficient to store the resource usages of all start times $\tau \in \Psi := \{\tau' \in H \mid \tau' \in \mathcal{U}_k \vee \tau' + p_i \in \mathcal{U}_k\}$ with $\mathcal{U}_k := \mathcal{U}^s_k \cup \mathcal{U}^e_k \cup \{0, \bar{d}\}$ to be able to calculate the resource usage for any start time $\tau \in H$. In order to determine the resource usage of any resource $k \in \mathcal{R}_i$ by activity $i \in V$ for any start time $\tau \in H$, a list $[r^u_{ik}(t)]$ which contains the resource usages of all $t \in \Psi$ ordered by increasing values of t is determined which can be used to calculate the resource usage for any start time $\tau \in H$ by

$$r^u_{ik}(\tau) := r^u_{ik}(\tau') + sgn\,(r^u_{ik}(\tau'') - r^u_{ik}(\tau')) \cdot (\tau - \tau')$$

with $\tau' := \max\{\sigma \in \Psi \mid \sigma \leq \tau\}$ and $\tau'' := \min\{\sigma \in \Psi \mid \sigma \geq \tau\}$. It is easy to verify that the number of start times in Ψ is given by $\mathcal{O}(\mathcal{I}_k)$ with $\mathcal{I}_k$ as the number of components in Π_k.

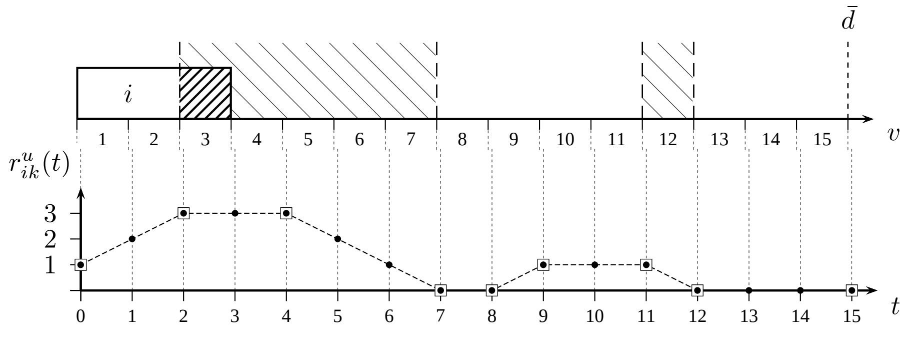

Figure 1: Course of the resource usage over all times of the planning horizon

In order to illustrate the concept of storing the resource usage for a subset of times of the planning horizon Ψ in a list $[r^u_{ik}(t)]$, we discuss the example illustrated in Figure 1. The example shows the resource usage course of a resource $k \in \mathcal{R}_i$ by an activity $i \in V$ over all start times $\tau \in H$ with $\Pi_k = \{3, \ldots, 7, 12\}$ and $p_i = 3$. The resource usage $r^u_{ik}(\tau)$ of start time $\tau = 0$ and also the set Π_k are outlined by hatched areas. For this example, we get $\mathcal{U}_k = \mathcal{U}^s_k \cup \mathcal{U}^e_k \cup \{0, \bar{d}\} = \{2, 11\} \cup \{7, 12\} \cup \{0, 15\} = \{0, 2, 7, 11, 12, 15\}$ so that $\Psi = \{\tau' \in H \mid \tau' \in \mathcal{U}_k\} \cup \{\tau' \in H \mid \tau' + p_i \in \mathcal{U}_k\} = \{0, 2, 7, 11, 12, 15\} \cup \{4, 8, 9, 12\} = \{0, 2, 4, 7, 8, 9, 11, 12, 15\}$ follows directly. The start times $\tau \in \Psi$, for which, as explained before, it is sufficient to store the resource usage in a list $[r^u_{ik}(t)]$, are marked by squares in Figure 1.

Based on the previous explanations, we can now state more formally that the planning horizon H can be subdivided in $\mathcal{O}(\mathcal{I}_k)$ time intervals $[a,b]$ with $\Delta^u_{ik}(\tau) = \Delta^u_{ik}(\tau+1)$ for all $\tau \in [a,b[\cap\mathbb{Z}$ for any activity $i \in V$ and resource $k \in \mathcal{R}_i$. In the following we say that activity $i \in V$ has a constant resource usage course of resource $k \in \mathcal{R}_i$ on any time interval $[a,b]$ if $\Delta^u_{ik}(\tau) = \Delta^u_{ik}(\tau+1)$ for all $\tau \in [a,b[\cap\mathbb{Z}$. Considering all resources $k \in \mathcal{R}_i$ of an activity $i \in V$, it follows directly that there is a polynomially bounded number of time intervals $[a,b]$ over the whole planning horizon H for activity $i \in V$ with a constant resource usage course of all resources $k \in \mathcal{R}_i$. As an example for a time interval with a constant course of all resource usages of an activity, see the left part of Figure 2. In what follows, we establish conditions for time intervals $[a,b]$ in scheduling set Θ_i with a constant resource usage course over all resources of activity $i \in V$ which, as shown later on, are used in Algorithm 3 to calculate T_i.

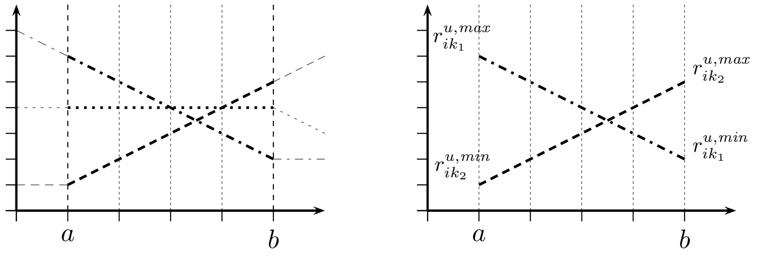

Figure 2: Time interval with a constant course of all resource usages

Lemma 4. *Let $\Theta_i \subseteq H$ be some scheduling set determined in Algorithm 1 for any not currently scheduled activity $i \in \bar{C}$ and let $\{a, a+1, \ldots, b\} \subseteq \Theta_i$ with $a < b$ and $\Delta^u_{ik}(\tau') = \Delta^u_{ik}(a)$ for all $\tau' \in [a,b[\cap\mathbb{Z}$ and all $k \in \mathcal{R}_i$ be given. Then any start time $\tau \in \Theta_i$ satisfies*

$$\nexists \tau' \in [a,b] \cap \mathbb{Z} : r^u_{ik}(\tau) \geq r^u_{ik}(\tau') \text{ for all } k \in \mathcal{R}_i \tag{1}$$

if and only if at least one of the conditions

$$\exists k \in \mathcal{R}_i : r^u_{ik}(\tau) < r^{u,min}_{ik} \tag{2}$$

$$\exists (k_1,k_2) \in \mathcal{R}^- \times \mathcal{R}^+ : r^u_{ik_1}(\tau) + r^u_{ik_2}(\tau) < r^{u,min}_{ik_1} + r^{u,min}_{ik_2} + b - a \tag{3}$$

with $r^{u,min}_{ik} := \min\{r^u_{ik}(\tau) \mid \tau \in [a,b] \cap \mathbb{Z}\}$, $\mathcal{R}^- := \{k \in \mathcal{R}_i \mid \Delta^u_{ik}(a) < 0\}$ and $\mathcal{R}^+ := \{k \in \mathcal{R}_i \mid \Delta^u_{ik}(a) > 0\}$ is met.

Proof. First, we show that (1) is satisfied for any $\tau \in \Theta_i$ if at least one of the conditions (2) or (3) is met. Assume that (2) is satisfied. Then condition (1) follows directly, since there is at least one resource $k \in \mathcal{R}_i$ with $r^u_{ik}(\tau) < r^u_{ik}(\tau')$ for all $\tau' \in [a,b] \cap \mathbb{Z}$. Next we suppose that (3) is satisfied while (2) is not met by start time τ. It follows $r^u_{ik_1}(\tau) + r^u_{ik_2}(\tau) < r^{u,max}_{ik_1} + r^{u,min}_{ik_2}$ with $r^{u,max}_{ik_1} := \max\{r^u_{ik_1}(\tau) \mid \tau \in [a,b] \cap \mathbb{Z}\} = r^{u,min}_{ik_1} + b - a$ for at least one pair $(k_1,k_2) \in \mathcal{R}^- \times \mathcal{R}^+$, which can easily be derived from the right part of Figure 2. Together with $r^u_{ik_2}(\tau) \geq r^{u,min}_{ik_2}$ we get $r^u_{ik_1}(\tau) < r^{u,max}_{ik_1}$, which implies $\exists \tau' \in [a+1,b] \cap \mathbb{Z}$ with $r^u_{ik_1}(\tau) = r^u_{ik_1}(\tau')$. Since $r^u_{ik_1}(\tau'') + r^u_{ik_2}(\tau'') = r^{u,min}_{ik_1} + r^{u,min}_{ik_2} + b - a =: c_{k_1k_2}$ for all $\tau'' \in [a,b] \cap \mathbb{Z}$ (see Figure 2), we get $r^u_{ik_1}(\tau) = r^u_{ik_1}(\tau') = c_{k_1k_2} - r^u_{ik_2}(\tau')$ which leads to $r^u_{ik_2}(\tau) < r^u_{ik_2}(\tau')$ with (3). Finally, we can state that (1) is met, since $r^u_{ik_1}(\tau) = r^u_{ik_1}(\tau') < r^u_{ik_1}(\tau'')$ for all $\tau'' \in [a,\tau'[\cap\mathbb{Z}$ and $r^u_{ik_2}(\tau) < r^u_{ik_2}(\tau') \leq r^u_{ik_2}(\tau'')$ for all $\tau'' \in [\tau',b] \cap \mathbb{Z}$.

In the next step, we prove that (1) is not satisfied if neither (2) nor (3) is met. For this, let both conditions (2) and (3) not be satisfied and $r^u_{ik}(\tau) \geq r^{u,max}_{ik}$ be given for all $k \in \mathcal{R}_i$. Then (1) is not fulfilled, since $r^u_{ik}(\tau) \geq r^u_{ik}(a)$ for all $k \in \mathcal{R}_i$. Consequently, it remains to consider that there is at least one resource $k_1 \in \mathcal{R}^-$ with $r^u_{ik_1}(\tau) < r^{u,max}_{ik_1}$. Since this implies that there is at least one time $\tau' \in [a+1,b] \cap \mathbb{Z}$ with $r^u_{ik_1}(\tau) = r^u_{ik_1}(\tau')$, equivalent to the previous descriptions, we get $r^u_{ik_2}(\tau) \geq r^u_{ik_2}(\tau')$ for each $k_2 \in \mathcal{R}^+$. Now, we consider time $\bar{\tau} := \max\{\tau' \in [a+1,b] \cap \mathbb{Z} \mid \exists k \in \mathcal{R}^- : r^u_{ik}(\tau) = r^u_{ik}(\tau')\}$, for which $r^u_{ik_2}(\tau) \geq r^u_{ik_2}(\bar{\tau})$ for all $k_2 \in \mathcal{R}^+$ and $r^u_{ik_1}(\tau) \geq r^u_{ik_1}(\bar{\tau})$ for all $k_1 \in \mathcal{R}^-$ can directly be followed. Finally with $r^u_{ik}(\tau) \geq r^{u,min}_{ik}$ for all $k \in \mathcal{R}_i$ we can derive that (1) is not satisfied for start time τ since $r^u_{ik}(\tau) \geq r^u_{ik}(\bar{\tau})$ for all $k \in \mathcal{R}_i$. □

In the following we describe the second procedure to calculate T_i for any scheduling set Θ_i which is outlined in Algorithm 3. As the first procedure, Algorithm 3 starts with an empty reduced scheduling set T_i which is complemented in the course of the algorithm. In each iteration, the procedure operates on a subset Θ of the scheduling set

Algorithm 3: Reduced scheduling set (v2)

```
Input: Scheduling set $\Theta_i$
Output: Reduced scheduling set $T_i$
1  $T_i := \emptyset \qquad \Theta := \Theta_i$
2  while $\Theta \neq \emptyset$ do
3      $a := \min \Theta \qquad b := a \qquad \mathcal{R}^- := \emptyset \qquad \mathcal{R}^+ := \emptyset$
4      if $a \neq e^{\Theta_i}(a)$ then
5          $\mathcal{R}^- := \{k \in \mathcal{R}_i \mid \Delta^u_{ik}(a) < 0\}$
6          if $\mathcal{R}^- \neq \emptyset$ then
7              $\mathcal{R}^+ := \{k \in \mathcal{R}_i \mid \Delta^u_{ik}(a) > 0\}$
8              forall $k \in \mathcal{R}_i$ do
9                  $b_{ik} := \max\{\tau \mid \Delta^u_{ik}(\tau') = \Delta^u_{ik}(a)$ for all $\tau' \in [a, \tau[ \cap \mathbb{Z}\}$
10             $b := \min\big(\min_{k \in \mathcal{R}_i} b_{ik}, e^{\Theta_i}(a)\big)$
11     forall $k \in \mathcal{R}_i$ do
12         $r^{u,min}_{ik} := \min\{r^u_{ik}(\tau) \mid \tau \in [a, b] \cap \mathbb{Z}\}$
13     $T_i := T_i \cup ([a, b] \cap \Theta) \qquad \Theta := \Theta \setminus \{a, a+1, \dots, b\} \qquad \Theta' := \emptyset$
14     forall $k \in \mathcal{R}_i : r^{u,min}_{ik} > 0$ do
15         $\Theta' := \Theta' \cup \{\tau \in \Theta \mid r^u_{ik}(\tau) < r^{u,min}_{ik}\}$
16     forall $(k_1, k_2) \in \mathcal{R}^- \times \mathcal{R}^+$ do
17         $c := r^{u,min}_{ik_1} + r^{u,min}_{ik_2} + b - a$
18         $\Theta' := \Theta' \cup \{\tau \in \Theta \mid r^u_{ik_1}(\tau) + r^u_{ik_2}(\tau) < c\}$
19     $\Theta := \Theta'$
20 return $T_i$
```

Θ_i, where Θ equals Θ_i at the start of the algorithm. In the following, we call a set $I := \{s, s+1, \dots, e\} \subseteq \Theta_i$ a component of Θ_i exactly if $s-1, e+1 \notin \Theta_i$ with s and e as the start and the end time of component I, respectively. Furthermore, we denote by $e^{\Theta_i}(\tau)$ the end time of the component I in Θ_i corresponding to start time τ, i.e., $\tau \in I$. In each iteration of Algorithm 3, the variables a and b are determined which represent the start and the end of a time interval $[a, b]$ which is completely enclosed in some component I of Θ_i, which means $[a, b] \cap \mathbb{Z} \subseteq I$. First, a is set to the start time of the first component in Θ_i, where in case that the end time of the first component in Θ_i is greater than the start time ($a \neq e^{\Theta_i}(a)$), the set of all resources $k \in \mathcal{R}_i$ with $\Delta^u_{ik}(a) < 0$ is determined ($\mathcal{R}^-$). If $\mathcal{R}^- \neq \emptyset$, the set of all resources $k \in \mathcal{R}_i$ with $\Delta^u_{ik}(a) > 0$ ($\mathcal{R}^+$) is established as well, followed by the calculation of the greatest possible value b_{ik} for each resource $k \in \mathcal{R}_i$ ensuring a constant resource usage course on time interval $[a, b_{ik}]$. Accordingly with $b := \min\big(\min_{k \in \mathcal{R}_i} b_{ik}, e^{\Theta_i}(a)\big)$, a time interval $[a, b] \cap \mathbb{Z} \subseteq \Theta_i$ with a constant resource usage course over all resources $k \in \mathcal{R}_i$ on interval $[a, b]$ is determined. Otherwise, in case that either $a = e^{\Theta_i}(a)$ or $\mathcal{R}^- = \emptyset$, time interval $[a, b]$ with $b := a$ is considered for the following calculations instead. After the minimum resource usage $r^{u,min}_{ik} := \min\{r^u_{ik}(\tau) \mid \tau \in [a, b] \cap \mathbb{Z}\}$ on interval $[a, b]$ for each resource $k \in \mathcal{R}_i$ has been determined, start times $\{a, a+1, \dots, b\} \cap \Theta$ are added to T_i. Given $\Theta = \Theta^a_i := \{\tau \in \Theta_i \mid \tau \geq a \wedge \nexists \tau' \in [0, a[\cap \Theta_i : r^u_{ik}(\tau) \geq r^u_{ik}(\tau')$ for all $k \in \mathcal{R}_i\}$, which is ensured by the procedure in each iteration as shown later on, $[a, b] \cap \Theta = [a, b] \cap T_i$ (see line 13) can be derived as follows. While the relation is directly be given by the definition of T_i for $a = b$, the correctness can also be established for $a < b$, noticing that the procedure ensures that there is at least one resource $k \in \mathcal{R}^-$ with $r^u_{ik}(\tau') < r^u_{ik}(\tau)$ for each pair $(\tau, \tau') \in \{a, a+1, \dots, b\}^2$ with $\tau < \tau'$. In what follows, we prove the correctness of Algorithm 3, i.e., the procedure returns the reduced scheduling set of Θ_i. For this, based on the previous descriptions, it is sufficient to show that $\Theta' = \Theta^{b+1}_i$ at the end of each iteration since $\Theta^{b+1}_i = \Theta^\alpha_i$ with $\alpha = \min \Theta^{b+1}_i$ follows directly if $\Theta^{b+1}_i \neq \emptyset$, which implies $[b+1, \alpha[\cap T_i = \emptyset$. First, we can state $\tau \in \Theta' \Leftrightarrow \nexists \tau' \in [a, b] \cap \Theta_i : r^u_{ik}(\tau) \geq r^u_{ik}(\tau')$ for all $k \in \mathcal{R}_i$ with Θ' as the outcome of the operations in lines 13-18, where Lemma 4 is considered in case that $a < b$. Thus, given $\Theta = \Theta^a_i$, $\Theta' = \Theta^{b+1}_i$ can directly be derived. Finally, by mathematical induction, taking $\Theta_i = \Theta^a_i$ with $a = \min \Theta_i$ for $\Theta_i \neq \emptyset$ in the initialization step into account, we get the correctness of Algorithm 3.

The last part of this section is concerned with the time complexities of both algorithms to calculate the reduced scheduling set T_i. For this, we call a set $I := \{s, s+1, \dots, e\} \subseteq H \setminus \Upsilon$ a break of $\Upsilon \subseteq H$ exactly if $s-1, e+1 \in \Upsilon$, where we denote by $\mathcal{B}^\Upsilon$ the number of breaks in Υ. In compliance with set Θ_i, we call set $I \subseteq \Upsilon$ a component of Υ

precisely if $s-1, e+1 \notin \Upsilon$. It should be noted that each set $\Upsilon \subseteq H$ can be stored in memory by a list containing the start and the end times of all components in an increasing order, which is used for the sets Θ_i, Θ and T_i in the following. First, we take a look on the time complexity of Algorithm 2. Noticing that the number of start times in Θ_i is bounded from above by $\bar{d}$, where each start time is compared with at most all start times before, we can easily derive a maximal number of $\mathcal{O}(\bar{d}^2)$ comparisons between pairs of start times. Since for each comparison at most $|\mathcal{R}|$ resources have to be considered, we get a time complexity of $\mathcal{O}(\bar{d}^2|\mathcal{R}|)$ for the comparison of the resource usages between all start times. Furthermore, by using list $[r^u_{ik}(t)]$ to determine the resource usage for each start time $t \in \Theta_i$ and by storing the resource usages of all start times which are added to T_i, we get a time complexity of $\mathcal{O}(\bar{d}^2|\mathcal{R}|+\mathcal{I})$ for Algorithm 2 with $\mathcal{I}$ as the sum of the number of components $\mathcal{I}_k$ over all resources $k \in \mathcal{R}$. Since $\bar{d} \geq \mathcal{I}_k$ and thus $\bar{d}|\mathcal{R}| \geq \mathcal{I}$ is given, finally we can state a time complexity of $\mathcal{O}(\bar{d}^2|\mathcal{R}|)$ for Algorithm 2. In order to determine the time complexity of Algorithm 3, we assume Θ_λ to be the set Θ at the start of any iteration $\lambda \in \mathbb{Z}_{\geq 0}$. First we consider the operations in lines 14-18. For both loops, using lists $[r^u_{ik}(t)]$ for all resources $k \in \mathcal{R}_i$ and by storing Θ by the start and the end times of all components, we can derive time complexities of $\mathcal{O}(|\mathcal{R}|\mathcal{B}^{\Theta_\lambda}+\mathcal{I})$ and $\mathcal{O}(|\mathcal{R}|^2\mathcal{B}^{\Theta_\lambda}+|\mathcal{R}|\mathcal{I})$, respectively. Next, we can observe that the number of iterations is $\mathcal{O}(\mathcal{B}^{\Theta_i}+\mathcal{I})$ by noticing that start time a, considering two consecutive iterations, is either assigned to a following component in Θ_i or skips at least one element in any list $[r^u_{ik}(t)]$ of some resource $k \in \mathcal{R}_i$. It should be noted, that in case $a \neq e^{\Theta_i}(a)$ and $\mathcal{R}^- = \emptyset$, start time a also skips at least one element in some list $[r^u_{ik}(t)]$ since start time $\tau = \min \Theta_i^{b+1} - 1$ has to satisfy $\Delta^u_{ik}(\tau) < 0$ for some $k \in \mathcal{R}_i$. Taking into account that start time a is monotonically increasing over all iterations, $e^{\Theta_i}(a)$ can be determined over all iterations by considering each start and each end time of a component in Θ_i at most once. The same applies to the elements of all lists $[r^u_{ik}(t)]$ to determine sets $\mathcal{R}^-$, $\mathcal{R}^+$, values b_{ik} and also the minimum resource usage $r^{u,min}_{ik}$ on some interval $[a,b]$ for all $k \in \mathcal{R}_i$. Conclusively, observing that T_i is complemented in each iteration in $\mathcal{O}(\mathcal{B}^{\Theta_\lambda})$ times, we get a time complexity of $\mathcal{O}(|\mathcal{R}|^2\mathcal{B}^{\Theta_\lambda}+|\mathcal{R}|\mathcal{I})$ for each iteration. Noticing, that the maximal number of breaks added by the procedures in lines 14-18 to set Θ_λ in each iteration is given by $\mathcal{O}(|\mathcal{R}|\mathcal{I})$, we can state that the maximal number of breaks to be considered in Θ_λ in each iteration is $\mathcal{O}((\mathcal{B}^{\Theta_i}+\mathcal{I})|\mathcal{R}|\mathcal{I})$. In conclusion, we get a time complexity of $\mathcal{O}((\mathcal{B}^{\Theta_i}+\mathcal{I})^2|\mathcal{R}|^3\mathcal{I})$ for Algorithm 3.

5 Improving techniques

In Watermeyer and Zimmermann (2020) it has already been shown for a relaxation-based branch-and-bound algorithm for the 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 in order to be able to use the consistency tests and lower bounds which have successively been applied in Watermeyer and Zimmermann (2020). As it will be shown later on in Sect. 5.3, the extension of the enumeration scheme is also used for techniques to prevent redundancies in the search tree.

For the extension of the enumeration scheme we establish a domain $W_i \subseteq H$ for the start time S_i of each activity $i \in V$, where 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 any not currently scheduled activity $i \in \bar{\mathcal{C}}$ is W-feasible exactly if augmentation $S^{\mathcal{C}\cup\{i\}}$ is W-feasible. Accordingly, 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 $j \in V$ can be 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 manner, 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 which 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, both with a time complexity of $\mathcal{O}(|V||E|(\mathcal{B}+1))$ with $\mathcal{B}$ as the total number of breaks in W. For details, we refer the reader to Watermeyer and Zimmermann (2020).

The extension of the enumeration scheme is given in Algorithm 4, where a start time restriction W is stored in addition for each enumeration node, 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 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 $t \in T_i$ which 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

Algorithm 4: Extended enumeration scheme

```
Input: Instance of problem RCPSP/max-$\pi$
Output: Set $\Phi$ of candidate schedules
1  Determine distance matrix $D = (d_{ij})_{i,j \in V}$
2  $ES_i := d_{0i}$, $LS_i := -d_{i0}$ for all $i \in V$
3  $W_i := \{ES_i, \ldots, LS_i\}$ for all $i \in V$
4  $\mathcal{C} := \{0\}$   $S := ES$
5  $\Omega := \{(\mathcal{C}, S, W)\}$   $\Phi := \emptyset$
6  while $\Omega \neq \emptyset$ do
7      Remove $(\mathcal{C}, S, W)$ from set $\Omega$
8      if $\mathcal{C} = V$ then
9          $\Phi := \Phi \cup \{S\}$
10     else
11         Select activity $i \in \bar{\mathcal{C}}$
12         $\Theta_i := \{\tau \in W_i \mid r_k^c(S^{\mathcal{C}}) + r_{ik}^c(\tau) \leq R_k$ for all $k \in \mathcal{R}_i\}$
13         Compute $T_i := ReducedSchedulingSet(\Theta_i)$
14         forall $t \in T_i$ do
15             $S'_i := t$   $S' := \min \widetilde{\mathcal{S}}_T(W, i, S'_i)$
               $W' := (W_j \setminus [0, S'_j[)_{j \in V}$
16             if $\exists j \in \mathcal{C} : S'_j > S_j$ then
17                 $\mathcal{C}' := \mathcal{C} \setminus \{j \in \mathcal{C} \mid S'_j > S_j\}$
18             else if $S'_i = t$ then
19                 $\mathcal{C}' := \mathcal{C} \cup \{i\}$
20             $\Omega := \Omega \cup \{(\mathcal{C}', S', W')\}$
21 return $\Phi$
```

$i \in \bar{\mathcal{C}}$ 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 $S' := \min \widetilde{\mathcal{S}}_T(W, i, t)$. Obviously, 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 $S'_j > S_j$. As a consequence, activity i can only be scheduled W-feasible at time t if $S'_i = t$. Finally, it should be mentioned that Algorithms 1 and 4 construct exactly the same search tree if no start time restriction with a break is present in the whole enumeration tree corresponding to Algorithm 4. This can directly be derived from $\min \widetilde{\mathcal{S}}_T(W, i, t) = (\max(S_j, t + d_{ij}))_{j \in V}$ with $S = (\min W_i)_{i \in V}$ in case that W with $W_i \neq \emptyset$ for all $i \in V$ does not contain any break.

In what follows, we prove the total correctness of the extended enumeration scheme which is closely related to the proof of the correctness for Algorithm 1. First, Theorem 2 states the completeness of the extended enumeration scheme, where Theorem 2 is based on Lemma 5 which represents a generalization of Lemma 1. Finally, taking into account that Lemma 3 also applies to Algorihm 4 and that each candidate schedule $S \in \Phi$ of Algorithm 4 is feasible, we can state the total correctness of the extended enumeration scheme.

Theorem 2. *Algorithm 4 generates all active schedules, i.e., $\Phi \supseteq \mathcal{AS}$.*

Proof. See the proof of Theorem 1. □

Lemma 5. *Let $S^f \in \mathcal{S}$ be any feasible schedule and $(\mathcal{C}, S, W)$ some node corresponding to the enumeration scheme of Algorithm 4 with $\mathcal{C} \neq V$, $S \leq S^f$, $S^f \in \mathcal{S}_T(W)$ and $r_{jk}^u(S_j) \leq r_{jk}^u(S_j^f)$ for all $j \in \mathcal{C}$ and all $k \in \mathcal{R}_j$. Then there is at least one direct descendant node $(\mathcal{C}', S', W')$ which fulfills the conditions $S' \leq S^f$, $S^f \in \mathcal{S}_T(W')$ and $r_{jk}^u(S'_j) \leq r_{jk}^u(S_j^f)$ for all $j \in \mathcal{C}'$ and all $k \in \mathcal{R}_j$.*

Proof. Equivalent to the proof of Lemma 1, based on Lemma 2, considering some activity $i \in \bar{\mathcal{C}}$ selected for the branching step, there is always a start time $t = \min\{\tau \in \Theta_i \mid r^u_{ik}(\tau) \leq r^u_{ik}(S^f_i)$ for all $k \in \mathcal{R}_i\} \in T_i$ with $t \leq S^f_i$ 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, $t \leq S^f_i$ and $S^f \in \mathcal{S}_T(W)$ imply $S' := \min \widetilde{\mathcal{S}}_T(W, i, t) \leq S^f$ and hence $S^f \in \mathcal{S}_T(W')$ with $W' := (W_j \setminus [0, S'_j[)_{j \in V}$. Furthermore, 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 ($\mathcal{C}' := \mathcal{C} \cup \{i\}$), some activities are unscheduled, or the number of scheduled activities remains unchanged. $\square$

5.1 Lower bounds

The calculation of lower bounds on the objective function value for minimization problems constitutes an integral part of a branch-and-bound algorithm. In this section, we present two lower bounds on the project duration which 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. This lower bound can be seen as an extension of the well-known critical path-based lower bound, where certain start times of activities are not allowed. The corresponding lower bound is given by $LB0^\pi := ES_{n+1}(W)$ with $ES(W) := \min \mathcal{S}_T(W)$ which can be calculated with a time complexity of $\mathcal{O}(|V||E|(\mathcal{B}+1))$.

The second lower bound LBD^π is determined in a destructive way, which means that a hypothetical upper bound d on the project duration is increased as long as it can be shown that 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 which has already been found in the course of the branch-and-bound procedure or $\bar{d} + 1$, otherwise. Let $[LB^d, UB^d]$ be the time interval which is considered in any iteration, then the binary search works as follows. First the upper bound $d := \lceil (LB^d + UB^d)/2 \rceil$ is set, followed by checking if d precludes any feasible solution. If this is the case, d can be rejected, which means that interval $[d+1, UB^d]$ can be investigated next, whereas otherwise, interval $[LB^d, d-1]$ is considered. The procedure to check if the assumption that $S_{n+1} \leq d$ contradicts the existence of any feasible solution is based on the calculation of the minimum resource consumptions of all resources over all activities of the project. For this, the latest W-feasible start time $LS^d_i(W) := LS_i(W, n+1, d)$ for each activity $i \in V$ is determined, where the total minimum resource consumption of any resource $k \in \mathcal{R}$ is given by

$$\underline{r}^c_k(W, d) := \sum_{i \in V_k} \underline{r}^c_{ik}(W, d),$$

with $\underline{r}^c_{ik}(W, d) := \min\{r^c_{ik}(\tau) \mid \tau \in W_i \cap [ES_i(W), LS^d_i(W)]\}$. In case that $\underline{r}^c_k(W, d) > R_k$ for at least one resource $k \in \mathcal{R}$, d is rejected, whereas otherwise it cannot be shown that d precludes any feasible solution. The time complexity of $\mathcal{O}(\log(\bar{d})(|V||E|(\mathcal{B}+1) + |\mathcal{R}|\mathcal{B} + |V|\mathcal{I}))$ for the calculation of LBD^π has been shown in Watermeyer and Zimmermann (2020).

5.2 Consistency tests

Consistency tests have already proven to be crucial for the performance of solution procedures for the RCPSP/π (Alvarez-Valdes et al., 2006, 2008, 2015) and the RCPSP/max-π (Watermeyer and Zimmermann, 2020). In what follows, we outline consistency tests which have already successively been applied for a relaxation-based branch-and-bound algorithm for the RCPSP/max-π in Watermeyer and Zimmermann (2020), where we refer the reader to this reference for further details.

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 $i \in V$. In order to evaluate the consistency tests, we use the term fixed point, where $W' := \gamma(W)$ is said to be the fixed point of some consistency test if either $W' = W$ or at least one start time restriction is empty, i.e., $W_i = \emptyset$.

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., Dorndorf et 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, where both tests are gathered under the term temporal-bound consistency test in this work.

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

One pass of the temporal-bound consistency test checks the conditions for each activity pair $(i,j) \in E$ exactly once. As it has been shown in Watermeyer and Zimmermann (2020), the fixed point of the temporal-bound consistency test can be determined by setting $W_i' := W_i \cap [ES_i(W,0,0), LS_i(W,0,0)]$ for all $i \in V$ with a time complexity of $\mathcal{O}(|V||E|(\mathcal{B}+1))$.

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

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

In Watermeyer and Zimmermann (2020) an algorithm has been developed to determine the fixed point of the temporal consistency test with a time complexity of $\mathcal{O}(|V|^2|E|(\mathcal{B}+1))$. It should be noted that the temporal consistency test dominates the temporal-bound consistency test in the sense that $W_i^t \subseteq W_i^b$ for all $i \in V$ with W^t and W^b as the fixed points of the temporal and the temporal-bound consistency test, respectively. In the remainder of this work, we call each start time $t \in W_i^t$ of some activity W-feasible.

Next, we deal with consistency tests which take the resource constraints $\sum_{i \in V} r_{ik}^c(S_i) \leq R_k$ for all $k \in \mathcal{R}$ of problem (P) into account, where the temporal constraints are only considered in some of them. All the following tests have in common that they consider each possible start time of some activity, where it is checked if the induced resource consumption of the activity and the minimum consumptions of all other activities of the project exceed the capacity of at least one resource. As it is shown later on, the consistency tests only differ in the way to calculate the minimum resource consumptions. First, we consider the so-called resource-bound consistency test which does not consider any temporal constraint at all. Accordingly, it is assumed that start time t of activity $i \in V$, which is checked by the consistency test, does not restrict the possible start times $\tau \in W_j$ of activity $j \in V \setminus \{i\}$. As a consequence, the minimum resource consumption of each activity $j \in V \setminus \{i\}$ is given by $r_{jk}^{c,min}(W) := \min\{r_{jk}^c(\tau) \mid \tau \in W_j\}$, so that the resource-bound consistency test is given by the condition

$$\exists k \in \mathcal{R}_i : r_{ik}^c(t) + \sum_{j \in V_k \setminus \{i\}} r_{jk}^{c,min}(W) > R_k$$

and the reduction rule $W_i := W_i \setminus \{t\}$. Thereby one pass can be conducted over all activities and their possible start times with a time complexity of $\mathcal{O}(|V|\mathcal{I} + |\mathcal{R}|\mathcal{B})$ as it has been shown in Watermeyer and Zimmermann (2020).

The last two consistency tests can be seen as extensions of the resource-bound consistency test in the sense that temporal constraints are used in addition to restrict the possible start times of the activities for the calculation of the minimum resource consumptions. The following consistency test makes use of the indirect time lags d_{ij} between the start times of activity i and all activities $j \in V \setminus \{i\}$, so that activity j can only be started at times $\tau \in W_j \cap [t + d_{ij}, t - d_{ji}]$. Consequently, the minimum resource consumption is given by $r_{ijkt}^{c,min}(W, D) := \min\{r_{jk}^c(\tau) \mid \tau \in W_j \cap [t + d_{ij}, t - d_{ji}]\}$ with distance matrix $D := (d_{ij})_{i,j \in V}$. The corresponding condition of the so-called D-interval consistency test for the reduction rule $W_i := W_i \setminus \{t\}$ can be stated by

$$\exists k \in \mathcal{R} : r_{ik}^c(t) + \sum_{j \in V_k \setminus \{i\}} r_{ijkt}^{c,min}(W, D) > R_k,$$

where one pass can be conducted with a time complexity of $\mathcal{O}(|V|^2\mathcal{I}^2 + |V||\mathcal{R}|\mathcal{B}^2)$ (Watermeyer and Zimmermann, 2020).

Finally, the last consistency test restricts the possible start times to calculate the minimum resource consumptions even more by taking breaks in the start time restrictions for the determination of the indirect minimum and maximum time lags into consideration. For this purpose, algorithms have been developed in Watermeyer and Zimmermann (2020) to determine the indirect minimum (maximum) time lag $\widetilde{d}_{ij}(W,t)$ ($\widehat{d}_{ij}(W,t)$) between the earliest (latest) W-feasible start time of some activity $j \in V$ and any W-feasible start time t of activity $i \in V$. By using these time lags, the minimum resource consumption of any activity $j \in V \setminus \{i\}$ can be calculated by

$$r^{c,min}_{ijkt}(W, \widetilde{D}, \widehat{D}) := \min\{r^c_{jk}(\tau) \mid \tau \in W^r_j\}$$

with $\widetilde{D}$ and $\widehat{D}$ as matrices to store the minimum and maximum time lags and $W^r_j := W_j \cap [t + \widetilde{d}_{ij}(W,t), t - \widehat{d}_{ij}(W,t)]$. In line with the consistency tests described before, one pass of the so-called W-interval consistency test is conducted over the possible start times of all activities, where the condition

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

is used for the reduction rule $W_i := W_i \setminus \{t\}$. As it is shown in Watermeyer and Zimmermann (2020), one pass of this test can be conducted with a time complexity of $\mathcal{O}(|V|^2\mathcal{I}^2 + |V|^2\mathcal{I}\mathcal{B} + |V||\mathcal{R}|\mathcal{B}^2)$.

5.3 Pruning the enumeration tree

In this section we present two techniques which are able to reduce the set of schedules which are explored by a node in the course of the enumeration scheme. For both methods, the branching step of Algorithm 4 is extended by a procedure which reduces the start time domain of the branching activity. It should be noted, that in contrast to methods which 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 what follows, in a first step we develop the two pruning techniques separately from each other. Afterwards, 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 which 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 use the first pruning technique, which we call 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 4, $W' := W$ 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'. To show that the correctness of Algorithm 4 remains if the UPT is used, it is sufficient to show that Lemma 5 can still be applied. For this, we extend the proof of Lemma 5 by taking into account that the UPT is used. Accordingly, we have to consider the additional operations in line 15 as described before. Let $t = \min\{\tau \in \Theta_i \mid r^u_{ik}(\tau) \leq r^u_{ik}(S^f_i) \text{ for all } k \in \mathcal{R}_i\} \in T_i$ with $t \leq S^f_i$ and $r^u_{ik}(t) \leq r^u_{ik}(S^f_i)$ for all $k \in \mathcal{R}_i$ be given. Then it can easily be verified that $S^f \in \mathcal{S}_T(W')$ after $W' := W$ 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. Equivalent to the proof of Lemma 5, $S' := \min \widetilde{\mathcal{S}}_T(W', i, t) \leq S^f$ and $S^f \in \mathcal{S}_T(W')$ with $W' := (W'_j \setminus [0, S'_j[)_{j \in V}$ can directly be derived, where all other points of the proof can be used without any change. Conclusively, Lemma 5 can still be applied to Algorithm 4 even if the UPT is used, where the proof of the total correctness is straightforward. Since the application of the UPT in Algorithm 4 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 can directly be derived from Lemma 5.

Corollary 1. *Let the UPT be used in Algorithm 4, let $S^f \in \mathcal{S}$ be any feasible schedule, and $(\mathcal{C}, S, W)$ some node corresponding to the enumeration scheme of Algorithm 4 with $\mathcal{C} \neq V$ and $S^f \in \mathcal{S}_T(W)$. Then there is at least one direct descendant node $(\mathcal{C}', S', W')$ which 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 which explores a part of the time-feasible region $\mathcal{S}_T$ which 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 the UPT is not applied.

The second pruning technique is based on the consideration of the resource usages of all times in the reduced scheduling set which 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 4, except for the setting of W'_i which 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 which 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 order to show that Lemma 5 can still be applied if the ULT is used, the proof of Lemma 5 can be extended in the same way as for the first pruning technique, where we only have to replace the setting of W_i'. Accordingly, it remains to show that $S_i^f \in W_i'$ after $W_i' := \{\tau \in W_i' \mid \nexists \tau' \in [0, t[\, \cap T_i : r_{ik}^u(\tau) \geq r_{ik}^u(\tau')$ for all $k \in \mathcal{R}_i\}$ is set with $t = \min\{\tau \in \Theta_i \mid r_{ik}^u(\tau) \leq r_{ik}^u(S_i^f)$ for all $k \in \mathcal{R}_i\}$. Since the assumption $S_i^f \notin W_i'$ would contradict that t equals $\min\{\tau \in \Theta_i \mid r_{ik}^u(\tau) \leq r_{ik}^u(S_i^f)$ for all $k \in \mathcal{R}_i\}$, we can state that Lemma 5 still applies to Algorithm 4 if the ULT is used. As a consequence, the total correctness of Algorithm 4 with the ULT is given as well.

In what follows, we investigate the application of both pruning techniques in Algorithm 4. For this, we consider two direct descendants $(\mathcal{C}', S', W')$ and $(\mathcal{C}'', S'', W'')$ of any 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, $W_i' \cap W_i'' = \emptyset$ can easily be verified. 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 the UPT and the ULT be used in Algorithm 4, let $S^f \in \mathcal{S}$ be any feasible schedule, and $(\mathcal{C}, S, W)$ some node corresponding to the enumeration scheme of Algorithm 4 with $\mathcal{C} \neq V$ and $S^f \in \mathcal{S}_T(W)$. Then there is exactly one direct descendant node $(\mathcal{C}', S', W')$ which fulfills the condition $S^f \in \mathcal{S}_T(W')$.*

From Corollary 2, 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, it can directly be derived that each candidate schedule is generated exactly once in the enumeration scheme if both pruning techniques UPT and ULT are used. Furthermore, we can state that each part of the time-feasible region is explored exactly once in the course of the enumeration scheme which can be seen as a prevention of any redundancy in the search tree.

6 Branch-and-bound algorithm

In this section we present the general framework of our branch-and-bound algorithm which 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 branch-and-bound algorithm which 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 branch-and-bound algorithm, 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 which 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. In the following we call this traversing strategy, which increases the diversification of investigated paths in the enumeration tree, scattered-path search (SPS). After some node has been chosen to be explored next, the branching strategy determines the activity which is 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 which 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, which 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 which have shown promising results in preliminary tests. First, we deal with priority rules which 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 (LST) rule with $\pi_i = LS_i$ and the slack time (ST) rule with $\pi_i = LS_i - ES_i$. For both priority rules we have also tested dynamic versions which take the start time restrictions and the best found solution UB into account which 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 the ST rule (STd$^{\mathrm{I}}$) which 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 (TS) rule with $\pi_i = |Reach^{\prec}(i)|$

and $Reach^{\prec}(i)$ as the set of all successors of activity $i \in V$ in $G^{\prec}$, the path following (PF) rule 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 (MRC) rule with $\pi_i = p_i \sum_{k \in \mathcal{R}_i} r^d_{ik}$. 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 $p_i r^d_{ik}$ by an activity in relation to the maximal remaining resource capacity $\bar{R}_k$ is considered with

$$\bar{R}_k := R_k - r^c_k(S^{\mathcal{C}}) - \sum_{i \in \bar{\mathcal{C}}} \underline{r}^c_{ik}(W, LS^{UB}_i(W)).$$

We have tested the following two different versions. The total maximal additional relative resource consumption (TMAR) rule with

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

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

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 which 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 nondecreasing 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 so far 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

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

can be seen as a combination of a priority value which is highly related to the TMAR rule and a penalty term which 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 branch-and-bound algorithm. In this work, all generated descendant nodes are explored in an order of nondecreasing lower bounds on the project duration (LB) which has shown to provide good results in computational experiments.

Before we discuss our branch-and-bound algorithm in more detail, we have to make some additional notes corresponding to the consistency tests. While until now we have considered all consistency tests separately from each other, it is common practice to apply different consistency tests iteratively until no constraint can be deduced anymore or the infeasibility of the considered search space is proven, i.e., the fixed point is reached. In our branch-and-bound algorithm we consider three different sets of consistency tests Γ^B, Γ^D, and Γ^W, where Γ^B contains the temporal-bound and the resource-bound consistency test, Γ^D the temporal-bound and the D-interval consistency test, and Γ^W the temporal and the W-interval consistency test. In order to increase the number of inconsistent start times for each set of consistency tests, the temporal constraint $S_{n+1} < UB$ with UB as the project duration of the best found solution in the course of the branch-and-bound procedure is established in addition. Furthermore, to restrict the computational effort, we have used an iteration limit α. In the following, we denote by $\gamma^\alpha_\beta(W)$ the start time restriction which results if all consistency tests in Γ^β are iteratively applied on W with an iteration limit α. 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 which are demanded by the activity, we use the notations $\gamma^\alpha_\beta(W)[\mathcal{R}]$ and $\gamma^\alpha_\beta(W)[\mathcal{R}_i]$, respectively. Furthermore, we use $\alpha = \infty$ to imply that the fixed point corresponding to set Γ^β is determined.

In what follows, we outline the framework of our branch-and-bound procedure which is given in Algorithm 5. 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 any 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 5, a preprocessing step is performed

Algorithm 5: Branch-and-bound algorithm

```
Input: Instance of problem RCPSP/max-$\pi$
Output: Optimal schedule $S^*$
1  Determine distance matrix $D = (d_{ij})_{i,j \in V}$
2  $ES_i := d_{0i}$, $LS_i := -d_{i0}$ for all $i \in V$
3  $W_i := \{ES_i, \dots, LS_i\}$ for all $i \in V$
4  Apply preprocessing on $W$
5  if $\mathcal{S}_T(W) = \emptyset$ then terminate ($\mathcal{S} = \emptyset$)
6  $LB^G := \min W_{n+1}$
7  $\mathcal{C} := \{0\}$   $S := \min \mathcal{S}_T(W)$   $LB := LB^G$
8  $\Omega := \{(\mathcal{C}, S, W, LB)\}$   $UB := \bar{d} + 1$
9  while $\Omega \neq \emptyset$ do
10     Remove $(\mathcal{C}, S, W, LB)$ from stack $\Omega$
11     if $LB < UB$ then
12         Apply consistency tests in $\Gamma^D$ or $\Gamma^W$ on $W$
13         if $\mathcal{S}_T^{UB}(W) \neq \emptyset$ then
14             Select activity $i \in \mathcal{E}$ and initialize $\Lambda := \emptyset$
15             $\Theta_i := \{\tau \in W_i \mid r^c_k(S^{\mathcal{C}}) + r^c_{ik}(\tau) \leq R_k$ for all $k \in \mathcal{R}_i\}$
16             $T_i := ReducedSchedulingSet(\Theta_i)$
17             while $T_i \neq \emptyset$ do
18                 Remove $t$ from set $T_i$ (according to the generation strategy)
19                 $S'_i := t$   $S' := \min \widetilde{\mathcal{S}}_T(W, i, S'_i)$   $W' := (W_j \setminus [0, S'_j[)_{j \in V}$
20                 Apply consistency tests in $\Gamma^B$ on $W'$
21                 if $\mathcal{S}_T^{UB}(W') \neq \emptyset$ then
22                     $S' := \min \mathcal{S}_T(W')$
23                     if $\exists j \in \mathcal{C} : S'_j > S_j$ then
24                         $\mathcal{C}' := \mathcal{C} \setminus \{j \in \mathcal{C} \mid S'_j > S_j\}$
25                     else if $S'_i = t$ then
26                         $\mathcal{C}' := \mathcal{C} \cup \{i\}$
27                     if $\mathcal{C}' = V$ then
28                         $S^* := S'$   $UB := S^*_{n+1}$   $T_i := T_i \setminus [t+1, \infty[$
29                     else
30                         Compute lower bound $LB'$
31                         if $LB' < UB$ then
32                             $\Lambda := \Lambda \cup \{(\mathcal{C}', S', W', LB')\}$
33         Put all nodes from $\Lambda$ on stack $\Omega$ (according to the ordering strategy)
34 if $UB = \bar{d} + 1$ then terminate ($\mathcal{S} = \emptyset$)
35 else return $S^*$
```

on the start time restriction W, for which we calculate the fixed point of set Γ^W considering all resources, which means that $W := \gamma^\infty_W(W)[\mathcal{R}]$ is set. In case that the preprocessing step cannot prove the infeasibility of the problem instance ($\mathcal{S}_T(W) \neq \emptyset$), 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 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 4, the scheduling set Θ_i and the reduced scheduling set T_i are calculated, where the start times in T_i are considered in an order depending on the generation strategy. Given some start time t which has been removed from T_i, t is established as the earliest start time of the branching activity, where it should be noted that line 19 of Algorithm 5 can be adapted as described in Sect. 5.3 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', where 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 which are greater than t are removed, noticing that they cannot generate any better feasible solution. Otherwise, the lower bound LB' is calculated for the descendant node which can either be given by $LB0^\pi(W')$ or $LBD^\pi(W, ES_{n+1}(W), UB - 1)$. 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 nonincreasing 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 5 returns an optimal solution S^*.

7 Performance analysis

In this section we evaluate the performance of our branch-and-bound algorithm. For this, we conduct computational experiments on different benchmark test sets with partially renewable resources. To provide a comprehensive investigation, we compare our procedure with all available branch-and-bound algorithms from the literature for the RCPSP/max-π and the RCPSP/π.

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, we compare with each other in this section, were all coded in C++ and compiled by the 64-bit Visual Studio 2017 C++-compiler.

7.1 General temporal constraints

In the first part of the performance analysis we compare our constructive branch-and-bound algorithm (CBB) with the relaxation-based branch-and-bound procedure (RBB) in Watermeyer and Zimmermann (2020) whose favorable performance has been shown by a comparison with the mixed-integer linear programming solver IBM CPLEX. To the best of our knowledge, the RBB represents the only branch-and-bound algorithm (BnB) for the RCPSP/max-π which is available in the open literature so far. For the comparison of the CBB with the RBB, we have conducted computational studies on a benchmark test set which covers instance sets with $n = 10, 20, 50, 100, 200$ real activities, all of them with 30 partially renewable resources. The benchmark test set UBO$^\pi$, which is available online[1], is an adaptation of the well-known benchmark test set UBO for the RCPSP/max which has been generated by the instance generator ProGen/max (Schwindt, 1996, 1998). As it is described in Watermeyer and Zimmermann (2020), the test sets for the RCPSP/max-π, which are denoted by UBOn^π in the following, are the result of a replacement of the renewable resources by partially renewable resources which are generated in accordance with the procedure in Schirmer (1999, Sect. 10).

Table 1 provides an overview of the settings we have used for the CBB in the computational experiments depending on the instance size. While the most terms in Table 1 are in line with Sect. 6, 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. It should be noted that Table 1 lists the settings which have shown the best balance between the number of instances which could be solved to optimality and whose solvability status remained open among all settings we have tested.

As it can be seen in Table 1, the restriction of the eligible set for the branching step in accordance with strict orders ($\prec_{\text{D}}, \prec_{\text{C}}$) is only beneficial for greater instances. Furthermore, the computational studies reveal that resource-based priority values are preferable for small instances to select the branching activity, whereas temporal-based or network-based priority values are better suited for greater instances. It is also worth mentioning that the 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 the 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.

Table 2 shows the performance of the CBB and the RBB, where it is distinguished between CBB1 and CBB2 to indicate if the first or the second version to calculate the reduced scheduling set T_i is applied (see Algorithms 2 and

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

	UBO10$^\pi$	UBO20$^\pi$	UBO50$^\pi$	UBO100$^\pi$	UBO200$^\pi$
Traversing strategy	DFS	SPS [2 s]	SPS [5 s]	SPS [5 s]	SPS [15 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 [30]
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^π	LBD^π	LBD^π	$LB0^\pi$	$LB0^\pi$
Pruning techniques	UPT	UPT	UPT+ULT	UPT+ULT	UPT

Table 1: Settings for the performance analysis

3). For the performance analysis, we have used a time limit of 300 seconds. The results for the RBB are taken from Watermeyer and Zimmermann (2020), where the RBB has been tested on the same workstation under the same conditions as the CBB. In the first column, Table 2 gives the number of instances for which the earliest start time schedule ES is not optimal (#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 instance set the number of instances for which an optimal solution is found and proven to be optimal (#opt), infeasibility is shown (#inf), a feasible solution is found whose optimality is not proven (#feas) or the solvability status remains open (#open). Finally, the last two columns show the average computational time over all instances which are solved to optimality ($\varnothing_{\text{opt}}^{\text{cpu}}$) and are shown to be infeasible ($\varnothing_{\text{inf}}^{\text{cpu}}$).

		#nTriv	#opt	#feas	#inf	#open	$\varnothing_{\text{opt}}^{\text{cpu}}$	$\varnothing_{\text{inf}}^{\text{cpu}}$
UBO10$^\pi$	CBB1	693	534	534	159	0	0.067 s	0.056 s
	CBB2		534	534	159	0	0.068 s	0.056 s
	RBB		534	534	159	0	0.040 s	0.004 s
UBO20$^\pi$	CBB1	621	537	581	40	0	7.846 s	0.702 s
	CBB2		535	581	40	0	7.354 s	0.721 s
	RBB		500	578	40	3	8.076 s	8.006 s
UBO50$^\pi$	CBB1	527	183	491	5	31	13.774 s	26.827 s
	CBB2		183	491	5	31	14.082 s	26.911 s
	RBB		145	486	3	38	8.022 s	0.279 s
UBO100$^\pi$	CBB1	484	85	472	0	12	14.307 s	–
	CBB2		84	472	0	12	14.527 s	–
	RBB		79	465	0	19	20.681 s	–
UBO200$^\pi$	CBB1	466	93	446	0	20	23.447 s	–
	CBB2		92	445	0	21	24.419 s	–
	RBB		79	466	0	0	28.271 s	–

Table 2: Performance of CBB and RBB (300 s)

First, we can observe that CBB1 dominates CBB2 over all instances, even though there is no great difference between both versions. Based on this, for all following investigations, we assume CBB to be conducted with Algorithm 2 to calculate the reduced scheduling set. Considering the performance of the CBB and the RBB, Table 2 shows a great dominance of the CBB for instance sets UBO20$^\pi$, UBO50$^\pi$, and UBO100$^\pi$, whereas the RBB is able to solve the test set UBO10$^\pi$ in less computational time. The results for instance set UBO200$^\pi$ are in some way unexpected with respect to the results for the smaller instances. While the CBB is able to solve more instances to optimality than the RBB, there are some instances for which the solvability status remains open which are shown to be feasible by the RBB. The reason for this may be suspected in the increase of the difficulty to determine if some start time of a branching activity could lead to a feasible solution with the increase of the number of activities in the project. It should be noted that this result gives an important implication for heuristic procedures concerned with the generation of feasible solutions for the RCPSP/max-π. Since the results for the test set UBO200$^\pi$ suggests that serial schedule-generation schemes which are based on the constructive procedure as described for the CBB might lack the ability to find feasible solutions for large instances, generation schemes based on the RBB should be taken into account as well for schedule-generation procedures.

In order to evaluate the quality of the best found solutions by the CBB for which the optimality could not be shown, Table 3 compares the project durations of the best solutions which have been found within a time limit of 300 seconds by the CBB with those of the RBB. 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 (#$^{\cup}_{\text{feas}}$) or by both procedures (#$^{\cap}_{\text{feas}}$), followed by the number of instances for which only the CBB (#$^{<}_{\text{feas}}$) or the RBB (#$^{>}_{\text{feas}}$) was able to find a feasible solution.

instance set	#$^{\cup}_{\text{feas}}$	#$^{\cap}_{\text{feas}}$	#$^{<}_{\text{feas}}$	#$^{>}_{\text{feas}}$	#$^{\cap,\text{nv}}_{\text{feas}}$	#$^{<}$	#$^{=}$	#$^{>}$	$\varnothing^{\Delta,\text{abs}}_{\text{RBB}}$	$\varnothing^{\Delta,\text{rel}}_{\text{RBB}}$
UBO20$^{\pi}$	581	578	3	0	79	37	32	10	-2.37	-1.55 %
UBO50$^{\pi}$	491	486	5	0	344	290	38	16	-16.87	-4.74 %
UBO100$^{\pi}$	472	465	7	0	390	243	31	116	-14.51	-2.42 %
UBO200$^{\pi}$	466	446	0	20	372	202	13	157	-7.06	0.49 %

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

As it can be seen for the test sets UBO20$^{\pi}$, UBO50$^{\pi}$, and UBO100$^{\pi}$ in Table 3, the CBB is able to find a feasible solution for more instances than the RBB, where in addition there is no instance for which only the RBB detects a feasible solution. In contrast, in accordance with the results in Table 2, the opposite is the case for the test set UBO200$^{\pi}$. In the second part of Table 3, the quality of the feasible solutions are 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 (#$^{\cap,\text{nv}}_{\text{feas}}$). These instances are subdivided into the number of instances with a better (#$^{<}$), an equal (#$^{=}$), or a worse (#$^{>}$) found solution by the CBB compared with the solution of the RBB. The last two columns are concerned with the average deviations of the project durations of the best found solutions by the CBB to those of the RBB, which are assumed to be given by S^{CBB}_{n+1} and S^{RBB}_{n+1}, respectively. In the first column, the average absolute deviation $\Delta^{\text{abs}}_{\text{RBB}} := S^{\text{CBB}}_{n+1} - S^{\text{RBB}}_{n+1}$ over all considered instances ($\varnothing^{\Delta,\text{abs}}_{\text{RBB}}$) is given, while the second column depicts the average relative deviation $\Delta^{\text{rel}}_{\text{RBB}} := \Delta^{\text{abs}}_{\text{RBB}}/S^{\text{RBB}}_{n+1}$ to the project duration of the best found solution by the RBB ($\varnothing^{\Delta,\text{rel}}_{\text{RBB}}$). The second part of Table 3 shows that the CBB can obtain over all instance sets for more instances a feasible solution with a better objective function value than the RBB, where it should be noted that the difference to the number of instances for which the RBB could find a better solution decreases with the increase in the instance size. Furthermore, the last two columns indicate a dominance of the CBB in the sense of a better quality of the found solutions on average for all instance sets, except for UBO200$^{\pi}$, for which a positive average relative deviation can be observed.

In order to illustrate the impact of the improving techniques on the performance of the CBB, Table 4 shows the results for test set UBO20$^{\pi}$ 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 the 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 for the branching step, in Table 4 the results of two different basic versions of the CBB are listed which 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 the CBB, where it can be seen that the calculation of the reduced scheduling set as well as all improving techniques enhance the performance of the CBB with a significant reduction in the total computing time t_{cpu}.

	#opt	#feas	#inf	#open	$\varnothing^{\text{cpu}}_{\text{opt}}$	$\varnothing^{\text{cpu}}_{\text{inf}}$	t_{cpu}
BnB (basic version Θ_i)	120	546	3	72	29.134 s	49.019 s	153,043 s
BnB (basic version T_i)	142	560	5	56	25.654 s	65.402 s	146,170 s
+Preprocessing	214	567	22	32	23.165 s	0.493 s	120,468 s
+LBD^{π}	222	567	22	32	24.408 s	1.010 s	118,541 s
+Consistency tests	343	576	23	22	10.667 s	0.363 s	80,167 s
+UPT	537	581	40	0	7.846 s	0.702 s	17,442 s

Table 4: Impact of components on the performance for instance set UBO20$^{\pi}$ (300 s)

7.2 Precedence constraints

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

The first benchmark test 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, where all of them have been 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 instances with a time limit of 300 seconds. For the CBB and the RBB we have used the settings of test set UBO20$^\pi$, except for test set P25π, for which we have conducted the computational tests on the CBB with the settings of test set UBO50$^\pi$. Table 5 shows that both the CBB and the RBB dominate the BOT over all instances, while only small differences can be observed between the CBB and the RBB, except that the RBB tend to show lower computing times.

		#nTriv	#opt	#feas	#inf	#open	$\varnothing^{\text{cpu}}_{\text{opt}}$	$\varnothing^{\text{cpu}}_{\text{inf}}$
	CBB		827	827	1281	0	0.056 s	0.054 s
P10π	RBB	2108	827	827	1281	0	0.007 s	0.007 s
	BOT		827	827	1281	0	0.023 s	0.023 s
	CBB		188	188	16	0	1.845 s	0.051 s
P15π	RBB	204	188	188	16	0	2.114 s	0.002 s
	BOT		181	181	16	7	2.727 s	0.036 s
	CBB		139	142	17	6	0.112 s	0.052 s
P20π	RBB	165	139	142	17	6	0.660 s	0.002 s
	BOT		136	139	16	10	3.974 s	2.187 s
	CBB		112	116	14	6	0.109 s	0.062 s
P25π	RBB	136	112	115	14	7	0.018 s	0.010 s
	BOT		105	111	11	14	0.465 s	25.457 s
	CBB		104	104	8	10	0.072 s	0.053 s
P30π	RBB	122	104	104	8	10	0.029 s	0.003 s
	BOT		98	104	3	15	2.095 s	0.196 s

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

The second benchmark set for the RCPSP/π was generated by Schirmer (1999) and has later been 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 (j10, j20, j30, j40), respectively, with 30 partially renewable resources. Later, Alvarez-Valdes et al. (2006, 2008) added a test set with 960 instances, each of them with 60 real activities (j60) and 30 partially renewable resources. It should be noted that 9 instances of test set j10 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 Table 6, the results of the computational tests on the Schirmer and Alvarez-Valdes instances are given with a time limit of 300 seconds. As for the Böttcher instances, we have used the settings of test set UBO20$^\pi$ for the CBB and the RBB for the computational experiments, with the only exception that the CBB was conducted with the settings of UBO50$^\pi$ for test set j60. Table 6 reveals that both the CBB and the RBB outperform the BOT for the Schirmer-Alvarez-Valdes benchmark set. Furthermore, Table 6 shows slightly better results for the CBB compared to those of the RBB.

		#nTriv	#opt	#feas	#inf	#open	$\varnothing^{\text{cpu}}_{\text{opt}}$	$\varnothing^{\text{cpu}}_{\text{inf}}$
	CBB		803	803	5	0	0.063	0.055
j10	RBB	808	803	803	5	0	0.060	0.052
	BOT		802	802	5	1	0.171	0.041
	CBB		563	565	0	0	0.906	–
j20	RBB	565	564	565	0	0	1.785	–
	BOT		509	561	0	4	6.436	–
	CBB		431	453	0	0	3.189	–
j30	RBB	453	427	453	0	0	3.717	–
	BOT		345	435	0	18	5.573	–
	CBB		347	386	0	0	5.386	–
j40	RBB	386	341	386	0	0	4.103	–
	BOT		261	363	0	23	4.376	–
	CBB		269	346	0	0	8.476	–
j60	RBB	346	268	346	0	0	2.172	–
	BOT		186	309	0	37	2.502	–

Table 6: Performance on the Schirmer-Alvarez-Valdes benchmark set (300 s)

8 Conclusions

We have presented a branch-and-bound algorithm (BnB) for the resource-constrained project scheduling problem with partially renewable resources and general temporal constraints (RCPSP/max-π). The enumeration of the BnB is based on a serial schedule-generation scheme with an unscheduling step. For the basic procedure of our BnB, we have shown how domains for the start times can be included to apply improving techniques from the literature. Moreover, we have developed further techniques to reduce the enumeration tree which are able to prevent redundancies in the course of the enumeration.

In a comprehensive performance analysis we have compared our exact solution procedure with all BnB which are available in the open literature for the RCPSP/max-π and the RCPSP/π. The computational experiments could reveal a great dominance of our BnB for instances with up to 100 activities for the RCPSP/max-π. Furthermore, the favorable performance could also be confirmed for instances of the RCPSP/π, where especially a great dominance over the only available BnB for the RCPSP/π could be demonstrated.

As the computational experiment has shown, the performance of a BnB for the 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, where a particular focus should be put on the prevention of redundancies and on an efficient integration of consistency tests.

References

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

Alvarez-Valdes, R., Crespo, E., Tamarit, J. M., and 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., and 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., and Villa, F. (2015). Partially renewable resources. In Schwindt, C. and Zimmermann, J., editors, *Handbook on Project Management and Scheduling Vol.1*, pages 203–227. Springer, Cham.

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., and Salewski, F. (1999). Project scheduling under partially renewable resource constraints. *Management Science*, 45(4):543–559.

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

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

Dorndorf, U., Pesch, E., and 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.

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

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

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

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.

Talbot, F. B. and 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. and 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.