OR Spectrum (2020) 42:427–460
https://doi.org/10.1007/s00291-020-00583-z

REGULAR ARTICLE

A branch-and-bound procedure for the resource-constrained project scheduling problem with partially renewable resources and general temporal constraints

Kai Watermeyer[1] · Jürgen Zimmermann[1]

Received: 3 September 2019 / Accepted: 19 February 2020 / Published online: 13 April 2020

Abstract
In this paper, we consider the resource-constrained project scheduling problem with partially renewable resources and general temporal constraints. For the first time, the concept of partially renewable resources is embedded in the context of projects with general temporal constraints. While partially renewable resources have already broadened the area of applications for project scheduling, the extension by general temporal constraints allows to consider even more relevant aspects of real projects. We present a branch-and-bound procedure for the problem with the objective to minimize the project duration. To improve the performance of the solution procedure, new consistency tests, lower bounds, and dominance rules are developed. Furthermore, new temporal planning procedures, based on forbidden start times of activities, are presented which can be used for any project scheduling problem with general temporal constraints independent on the considered resource type. In a performance analysis, we compare our branch-and-bound procedure with the mixed-integer linear programming solver IBM CPLEX 12.8.0 on adaptations of benchmark instances from the literature. In addition, we compare our solution procedure with the only available branch-and-bound procedure for partially renewable resources. The results of the computational experiments prove the efficiency of our branch-and-bound procedure.

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

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

Jürgen Zimmermann
juergen.zimmermann@tu-clausthal.de

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

1 Introduction

This paper is concerned with the resource-constrained project scheduling problem with partially renewable resources (RCPSP/π) extended by general temporal constraints which to the best of our knowledge has not been treated in the open literature so far. The concept of partially renewable resources has already proved to be useful to model constraints occurring in different real applications. The first usage of this resource type can be found in Drexl et al. (1993) for a course scheduling problem. Further examples are given in Drexl and Salewski (1997) for school timetabling, in Bartsch et al. (2006) and Knust (2010) for sports scheduling, in Briskorn and Fliedner (2012) for container transshipment, and in Okubo et al. (2015) for machine scheduling. The availability of a partially renewable resource is limited to an arbitrary set of periods. Accordingly, a partially renewable resource is only consumed by an activity in the periods the resource is available and the activity is in process. In the field of project scheduling, it is well known that partially renewable resources generalize the concepts of renewable resources with time-varying capacities and nonrenewable resources. Therefore, the usage of partially renewable resources in project scheduling problems opens new application areas, where the modeling of labor regulations in staff scheduling appears as one of the most promising fields.

Different approximation methods have been considered for the RCPSP/π. Böttcher et al. (1999) and Schirmer (1999) have both proposed a schedule-generation scheme where in Schirmer (1999) local search procedures have been investigated in addition. Furthermore, the works of Alvarez-Valdes et al. (2006, 2008) and Alvarez-Valdes et al. (2015) are dedicated to a GRASP and a scatter search algorithm for the RCPSP/π.

An alternative approach to extend the classical resource-constrained project scheduling problem (RCPSP) is given by the consideration of general temporal constraints (RCPSP/max) which has been studied thoroughly in the literature but has not yet been treated with partially renewable resources so far. Therefore, it seems promising to combine both extensions which we denote by RCPSP/max-π in the following. For an extensive overview of applications for general temporal constraints, we refer the reader to Neumann and Schwindt (1995) and Neumann et al. (2003).

For both extensions, branch-and-bound procedures have been developed. In Böttcher et al. (1999), the only branch-and-bound procedure for the RCPSP/π based on the work of Talbot and Patterson (1978) can be found. All other procedures in contrast are dedicated to the RCPSP/max given in Bartusch et al. (1988), De Reyck and Herroelen (1998), Schwindt (1998a, c), Fest et al. (1999), Dorndorf et al. (2000b), and Bianco and Caramia (2012). Since none of these procedures can directly be adapted to solve the RCPSP/max-π, the need for a new concept tackling the RCPSP/max-π is evident.

In this paper, we present a branch-and-bound procedure for the RCPSP/max-π complemented by efficient procedures to improve the performance. The following section describes the RCPSP/max-π formally. Section 3 covers the enumeration scheme of the branch-and-bound procedure and Sect. 4 presents temporal planning procedures which are used for the consistency tests concerned with in Sect. 5. Sections 6 and 7 are dedicated to lower bounds and dominance rules, respectively. In Sect. 9, the branch-

and-bound procedure is discussed, and Sect. 10 provides an experimental performance analysis. Finally, conclusions are presented in Sect. 11.

2 Problem description

The RCPSP/max-π is given by a project consisting of n real activities and two fictitious activities 0 and $n+1$ which represent the start and the end of the project, respectively. Each activity $i \in V = \{0, 1, \ldots, n+1\}$ is assigned a non-interruptible processing time $p_i \in \mathbb{Z}_{\geq 0}$ and a resource demand $r_{ik}^d \in \mathbb{Z}_{\geq 0}$ for each partially renewable resource $k \in \mathcal{R}$, where all fictitious activities have neither a processing time nor a resource demand. Between pairs of activities $(i, j) \in E \subset V \times V$ general temporal constraints are given. For each activity pair $(i, j) \in E$, a time lag $\delta_{ij} \in \mathbb{Z}$ between the start times of activity i and j has to be hold, i.e., $S_j \geq S_i + \delta_{ij}$. It should be noted that negative time lags can be interpreted as maximum time lags. Besides the temporal constraints, resource capacities R_k of all partially renewable resources $k \in \mathcal{R}$ have to be taken into consideration. Each resource is defined on a subset of all periods of the planning horizon $\Pi_k \subseteq \{1, 2, \ldots, \bar{d}\}$ with $\bar{d}$ as the prescribed maximum project duration. It should be noted that in the literature, partially renewable resources are also defined in other ways by assigning multiple subsets of periods to each of them. In this paper, we use the so-called normalized formulation for partially renewable resources which is beneficial for theoretical issues (Böttcher et al. 1999). An activity $i \in V$ consumes r_{ik}^d units of resource $k \in \mathcal{R}$ in each period of Π_k the activity is in execution. In the following, we call the number of periods in Π_k an activity $i \in V$ with start time S_i is in execution, given by $r_{ik}^u(S_i) := |]S_i, S_i + p_i] \cap \Pi_k|$, the resource usage of resource $k \in \mathcal{R}$ by activity i. Accordingly, the consumption of a resource $k \in \mathcal{R}$ by an activity $i \in V$ which starts at time S_i can be stated by $r_{ik}^c(S_i) := r_{ik}^u(S_i) \cdot r_{ik}^d$, where the consumption by all activities of the project $r_k^c(S) := \sum_{i \in V} r_{ik}^c(S_i)$ must not exceed the capacity R_k. To ensure that each activity is not executed just during a part of a period, the start times are restricted to integral values, i.e., $S_i \in \mathbb{Z}_{\geq 0}$ for all $i \in V$.

The objective of the RCPSP/max-π is to assign each activity a start time, so that all temporal and resource constraints are satisfied and the project duration is minimized with a presumed start of the project at time 0 and a maximum project duration $\bar{d}$. In the following, a sequence of start times of all activities $S = (S_i)_{i \in V}$ with $S_i \in \mathbb{Z}_{\geq 0}$ for all $i \in V$ and $S_0 = 0$ is called a schedule, where S is said to be time-feasible, resource-feasible or feasible if it fulfills all temporal constraints, all resource constraints or all constraints, respectively. The problem RCPSP/max-π can thus be stated as follows:

$$\left.\begin{array}{lll} \text{Minimize} & f(S) = S_{n+1} & \\ \text{subject to} & S_j - S_i \geq \delta_{ij} & ((i,j) \in E) \\ & \sum\limits_{i \in V} r_{ik}^c(S_i) \leq R_k & (k \in \mathcal{R}) \\ & S_0 = 0 & \\ & S_i \in \mathbb{Z}_{\geq 0} & (i \in V). \end{array}\right\} \text{(P1)}$$

It should be noted that in order to ensure the compliance with the maximum project duration $\bar{d}$ of each time-feasible schedule, a temporal constraint between the start and the end of the project is established, i.e., $(n+1, 0) \in E$ with $\delta_{n+1,0} = -\bar{d}$. Accordingly, the start time of each activity is restricted to set $H := \{0, 1, \dots, \bar{d}\}$ representing all integral times of the planning horizon. In the following, the feasible region of problem (P1) is denoted by $\mathcal{S}$ with $\mathcal{OS} \subseteq \mathcal{S}$ as the set of all optimal schedules. In addition, regarding the temporal and resource constraints, the sets of all time-feasible schedules $\mathcal{S}_\mathrm{T}$ and all resource-feasible schedules $\mathcal{S}_\mathrm{R}$ are considered as well.

3 Enumeration scheme

In this section, the enumeration scheme of our branch-and-bound procedure is described, i.e., the way to generate a set $\Phi \subseteq \mathcal{S}$ containing at least one optimal solution for problem (P1). This scheme can be illustrated as a directed out-tree, where each node is assigned a so-called start time restriction W which is defined as follows:

Definition 1 A vector $W = (W_i)_{i \in V}$ with $W_i \subseteq \{0, 1, \dots, \bar{d}\}$ for all $i \in V$ and $W_0 = \{0\}$ is called a start time restriction with W_i as the start time restriction of activity $i \in V$.

In each node of the enumeration tree, the resource relaxation of problem (P1) is considered with additional constraints, restricting the possible start times of each activity $i \in V$ to the values contained in W_i, i.e., $S_i \in W_i$. Thus, the problem corresponding to an enumeration node can be stated by

$$\left.\begin{array}{lll} \text{Minimize} & f(S) = S_{n+1} & \\ \text{subject to} & S_j - S_i \geq \delta_{ij} & ((i,j) \in E) \\ & S_i \in W_i & (i \in V). \end{array}\right\} \text{(P2(W))}$$

In the following, the solution space of problem (P2(W)) is termed W-feasible region denoted by $\mathcal{S}_\mathrm{T}(W) := \{S \in \mathcal{S}_\mathrm{T} \mid S_i \in W_i \text{ for all } i \in V\}$ and each schedule $S \in \mathcal{S}_\mathrm{T}(W)$ is called W-feasible. In Sect. 4, we show that $\mathcal{S}_\mathrm{T}(W) \neq \emptyset$ has a unique minimal point, where we call a schedule $S \in \mathcal{S}_\mathrm{T}(W)$ the unique minimal point of $\mathcal{S}_\mathrm{T}(W)$ if there is no other schedule $S' \in \mathcal{S}_\mathrm{T}(W)$ with $S'_i \leq S_i$ for all $i \in V$. Furthermore, an algorithm is covered, able to determine $\min \mathcal{S}_\mathrm{T}(W)$ exactly if there is at least one W-feasible schedule, or to prove $\mathcal{S}_\mathrm{T}(W) = \emptyset$, otherwise.

The construction of the directed out-tree or rather the enumeration tree is outlined in Algorithm 1. At first, for each activity $i \in V$ the earliest and latest time-feasible start times ES_i and LS_i are determined by a label-correcting algorithm (see, e.g., Ahuja et al. 1993, Sect. 5.4) if there is at least one time-feasible schedule, i.e., $\mathcal{S}_\mathrm{T} \neq \emptyset$. Otherwise, the label-correcting algorithm proves $\mathcal{S}_\mathrm{T} = \emptyset$, so that Algorithm 1 terminates.

Algorithm 1: Enumeration scheme

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

1  Determine $ES$ and $LS$ with a label-correcting algorithm
2  if $\mathcal{S}_T = \emptyset$ then terminate
3  $W_i := \{ES_i, .., LS_i\} \quad \forall i \in V$
4  $\Omega := \{W\} \qquad \Phi := \emptyset$
5  while $\Omega \neq \emptyset$ do
6      Remove $W$ from set $\Omega$
7      $S := \min \mathcal{S}_T(W)$
8      if $S \in \mathcal{S}_R$ then
9          $\Phi := \Phi \cup \{S\}$
10     else
11         Select resource $k \in \mathcal{R}$ with $r_k^c(S) = \sum_{i \in V} r_{ik}^u(S_i) \cdot r_{ik}^d > R_k$
12         forall $i \in \{j \in V_k \mid r_{jk}^u(S_j) > 0\}$ do
13             $W' := W \qquad W_i' := W_i' \cap W_{ik}(r_{ik}^u(S_i) - 1)$
14             if $\mathcal{S}_T(W') \neq \emptyset$ then $\Omega := \Omega \cup \{W'\}$
15 return $\Phi$
```

In the following, we assume that $\mathcal{S}_{\mathrm{T}} \neq \emptyset$. At the beginning of the construction process, the start time restriction $W = (W_i)_{i \in V}$ with $W_i = \{\mathrm{ES}_i, \mathrm{ES}_i + 1, \ldots, \mathrm{LS}_i\}$ for all activities $i \in V$ is assigned to the root node and added to set Ω, which is used in the process for saving all enumeration nodes not explored yet. Additionally, set Φ which gathers all potentially optimal schedules generated in the process, is initialized to an empty set. It should be noted that problem (P2(W)) corresponding to the root node is equal to the resource relaxation of problem (P1), so that $\mathcal{S}_{\mathrm{T}}(W) = \mathcal{S}_{\mathrm{T}} \supseteq \mathcal{S}$ holds.

The main step of Algorithm 1 describes the generation of the enumeration tree. In each iteration, a start time restriction W is removed from set Ω and the corresponding problem (P2(W)) is solved, i.e., $S = \min \mathcal{S}_{\mathrm{T}}(W)$ is determined. In case that schedule S is feasible, i.e., $r_k^c(S) := \sum_{i \in V} r_{ik}^c(S_i) \leq R_k$ for all $k \in \mathcal{R}$, schedule S is added to set Φ, meaning that the corresponding node represents a leaf of the enumeration tree. Otherwise, there is at least one conflict resource $k \in \mathcal{R}$ with $r_k^c(S) > R_k$. In this case, the solution space $\mathcal{S}_{\mathrm{T}}(W)$ is decomposed based on a reduction in the permitted maximum resource usages of all activities $i \in V_k := \{i \in V \mid r_{ik}^d > 0\}$ consuming conflict resource $k \in \mathcal{R}$. In the following, we will use $\bar{u}_{ik}$ as the so-called resource usage bound of activity $i \in V$ for resource $k \in \mathcal{R}$, representing an upper bound for the resource usage, i.e., $r_{ik}^u(S_i) \leq \bar{u}_{ik}$, added during the enumeration process. Additionally, $W_{ik}(\bar{u}_{ik}) := \{\tau \in \{\mathrm{ES}_i, \mathrm{ES}_i + 1, \ldots, \mathrm{LS}_i\} \mid r_{ik}^u(\tau) \leq \bar{u}_{ik}\}$ is said to be the start time restriction of activity $i \in V$ induced by the resource usage bound $\bar{u}_{ik}$, comprising all time-feasible start times of activity $i \in V$ with a maximum resource usage of $\bar{u}_{ik}$. The following explanations for the decomposition of $\mathcal{S}_T(W)$ are based on Theorem 1 which implicates that no feasible schedule $S \in \mathcal{S}$ is excluded by the enumeration procedure. It should be noted that therefore the enumeration process is independent on the considered objective function.

Theorem 1 *Given $\sum_{i \in V} u_{ik} r^d_{ik} > R_k$ with $u_{ik} \in \{0, 1, \ldots, p_i\}$ for resource $k \in \mathcal{R}$, each feasible schedule $S \in \mathcal{S}$ fulfills $S_i \in W_{ik}(u_{ik} - 1)$ for at least one activity $i \in V_k$.*

Proof Let S be a feasible schedule with $S_i \notin W_{ik}(u_{ik} - 1)$ for all activities $i \in V_k$. It follows that $r^u_{ik}(S_i) \geq u_{ik}$ for all activities, so that $\sum_{i \in V_k} r^u_{ik}(S_i) \cdot r^d_{ik} \geq \sum_{i \in V_k} u_{ik} \cdot r^d_{ik} > R_k$ is given. Since this contradicts the assumption that S is feasible, the theorem is proven. □

In what follows, we describe the decomposition of $\mathcal{S}_T(W)$ in subsets for some conflict resource $k \in \mathcal{R}$ so that each activity $i \in V_k$ corresponds to one of them. Let W be the start time restriction of any node in the enumeration tree with schedule $S = \min \mathcal{S}_T(W) \notin \mathcal{S}_R$ and conflict resource $k \in \mathcal{R}$. Then, the decomposition of $\mathcal{S}_T(W)$ works as follows. Regarding Theorem 1, for each activity $i \in V_k$ consuming conflict resource $k \in \mathcal{R}$, the maximum resource usage $\bar{u}_{ik}$ is set to $r^u_{ik}(S_i) - 1$, meaning that all start times $t \in W_i$ with $r^u_{ik}(t) \geq r^u_{ik}(S_i)$ are removed from W_i. This is achieved by setting $W'_i := W_i \cap W_{ik}(r^u_{ik}(S_i) - 1)$ and $W'_j := W_j$ for all $j \in V \setminus \{i\}$ with $\mathcal{S}_T(W')$ as the decomposition part of $\mathcal{S}_T(W)$ corresponding to activity $i \in V_k$. If $\mathcal{S}_T(W') \neq \emptyset$, W' is added to Ω and explored in one of the following iterations. After the exploration of all enumeration nodes, i.e., $\Omega = \emptyset$, Algorithm 1 terminates with output Φ containing all generated feasible schedules in the process.

Finally, it should be mentioned that the completeness of Algorithm 1 can easily be derived from Theorem 1 where the correctness follows directly with $\Phi \subseteq \mathcal{S}$. As a consequence of Lemma 1, we can additionally state that the enumeration scheme terminates after finitely many iterations.

Lemma 1 *Algorithm 1 generates at most $|V|^{|V||\mathcal{R}|p^{\max}}$ enumeration nodes.*

Proof Since the permitted usage of a resource by an activity can be reduced at most $p^{\max} := \max_{i \in V} p_i$ times, the maximum depth of the enumeration tree is given by $|V||\mathcal{R}|p^{\max}$. Taking also into consideration, that in each decomposition step at most $|V|$ enumeration nodes can be added to Ω, we get a maximum number of $|V|^{|V||\mathcal{R}|p^{\max}}$ enumeration nodes. □

4 Temporal planning with start time restrictions

In this section, we discuss temporal planning procedures which represent the backbone of the consistency tests described in Sect. 5 and which are used in the enumeration scheme to determine for each enumeration node the minimal point of its corresponding feasible region. In the first part, we consider two algorithms which are able to determine the earliest and latest start times of all activities of the project, where a start time restriction and a lower or upper bound for the start time of some activity are taken into account. Based on these procedures, the second part is concerned with the calculation of minimum and maximum time lags between the start times of all activity pairs of the project.

4.1 Earliest and latest start times

The first procedure can be seen as a label-correcting algorithm which is able to determine the unique minimal point of $\widetilde{\mathcal{S}}_\mathrm{T}(W, \alpha, t_\alpha) := \{S \in \mathcal{S}_\mathrm{T}(W) \mid S_\alpha \geq t_\alpha\}$ if it exists or which shows otherwise that $\widetilde{\mathcal{S}}_\mathrm{T}(W, \alpha, t_\alpha) = \emptyset$. In the following, we denote by $\mathrm{ES}(W, \alpha, t_\alpha)$ the minimal point of $\widetilde{\mathcal{S}}_\mathrm{T}(W, \alpha, t_\alpha)$, where $\mathrm{ES}_i(W, \alpha, t_\alpha)$ represents the earliest W-feasible start time of activity i if activity $\alpha \in V$ is not started earlier than at time t_α. As described in the next section, the limitation of $\mathcal{S}_\mathrm{T}(W)$ by $S_\alpha \geq t_\alpha$ is used to determine the start-time-dependent indirect minimum time lags between all activity pairs of the project. It should be noted that Algorithm 2 comprises the calculation of $\min \mathcal{S}_\mathrm{T}(W)$ by setting $\alpha := 0$ and $t_\alpha := 0$, i.e., establishing the project start to be greater than 0.

In what follows, we describe Algorithm 2 for which we use

$$\widetilde{\delta}_{ij}(W, \nu_i) := \begin{cases} \min\{\tau \in W_j \,|\, \tau \geq \nu_i + \delta_{ij}\} - \nu_i, & \text{if } \{\tau \in W_j \,|\, \tau \geq \nu_i + \delta_{ij}\} \neq \emptyset \\ \delta_{ij}, & \text{otherwise} \end{cases}$$

to represent the minimum time lag between the lowest start time $\nu_j \in W_j$ of activity $j \in V$ which satisfies the minimum time lag to the tentative start time ν_i of activity $i \in V$ (if it exists). In the first step, Algorithm 2 checks conditions which imply that $\mathcal{S}_\mathrm{T} = \emptyset$. In case that none of these conditions is satisfied, the earliest time $t'_\alpha \geq t_\alpha$ in W_α is assigned to node α, the initial weights of all other nodes $i \in V \setminus \{\alpha\}$ are set to $\nu_i := -\infty$ and Q, which is implemented as a queue, is initialized by $Q := \{\alpha\}$. In each iteration of Algorithm 2, the weights of the direct successors $\mathrm{Succ}(i)$ of all nodes $i \in V$ are considered which have been added to Q in the previous iteration. In case that $\nu_i + \widetilde{\delta}_{ij}(W, \nu_i) > \nu_j$ is detected, the weight ν_j is set to $\nu_i + \widetilde{\delta}_{ij}(W, \nu_i)$. Since the weight of node j is increased, in the next iteration the node weights of all its direct successors have to be checked which is ensured by adding j to Q. If the updated weight of node j is greater than $\max W_j$, $\widetilde{\mathcal{S}}_\mathrm{T}(W, \alpha, t_\alpha) = \emptyset$ can be stated. Finally, in case that Algorithm 2 terminates with $Q = \emptyset$, schedule $\min \widetilde{\mathcal{S}}_\mathrm{T}(W, \alpha, t_\alpha)$ is determined or rather $\mathrm{ES}(W, \alpha, t_\alpha)$ is returned. The correctness of the algorithm and its time complexity is stated in Theorem 2.

Algorithm 2: Minimal point of $\widetilde{\mathcal{S}}_T(W,\alpha,t_\alpha)$

```
Input: Activity $\alpha \in V$ and $t_\alpha$
Output: $ES(W,\alpha,t_\alpha)$
if $\mathcal{S}_T = \emptyset \vee \exists i \in V : W_i = \emptyset \vee t_\alpha > \max W_\alpha$ then
    terminate $(\widetilde{\mathcal{S}}_T(W,\alpha,t_\alpha) = \emptyset)$
$t'_\alpha := \min\{\tau \in W_\alpha \mid \tau \geq t_\alpha\}$
forall $i \in V$ do
    $\nu_i := \begin{cases} t'_\alpha & \text{, if } i = \alpha \\ -\infty & \text{, otherwise} \end{cases}$
$Q := \{\alpha\}$
while $Q \neq \emptyset$ do
    Dequeue $i$ from $Q$
    forall $j \in Succ(i)$ do
        if $\nu_i + \widetilde{\delta}_{ij}(W,\nu_i) > \nu_j$ then
            $\nu_j := \nu_i + \widetilde{\delta}_{ij}(W,\nu_i)$
            if $j \notin Q$ then enqueue $j$ into $Q$
            if $\nu_j > \max W_j$ then
                terminate $(\widetilde{\mathcal{S}}_T(W,\alpha,t_\alpha) = \emptyset)$
return $(\nu_i)_{i\in V}$
```

In the following, we call a set $I := \{a, a+1, \dots, b\} \subset \mathbb{Z}$ a start time component of start time restriction W_i if and only if the conditions $I \subseteq W_i$ and $a-1, b+1 \notin W_i$ are satisfied. Otherwise, in case that $I \subseteq H \setminus W_i$ with $a-1, b+1 \in W_i$ is given, I is said to be a start time break of W_i, where the number of start time breaks in W_i and W is denoted by $\mathcal{B}_i$ and $\mathcal{B}$, respectively.

Theorem 2 *Algorithm* 2 *determines the unique minimal point of* $\widetilde{\mathcal{S}}_T(W,\alpha,t_\alpha)$ *or shows that* $\widetilde{\mathcal{S}}_T(W,\alpha,t_\alpha) = \emptyset$ *with a time complexity of* $\mathcal{O}(|V||E|(\mathcal{B}+1))$.

Proof Let $S \in \widetilde{\mathcal{S}}_T(W,\alpha,t_\alpha)$ be given. From $t'_\alpha := \min\{\tau \in W_\alpha \mid \tau \geq t_\alpha\} \leq S_\alpha$ and $\nu'_i + \widetilde{\delta}_{ij}(W,\nu'_i) \leq \nu''_i + \widetilde{\delta}_{ij}(W,\nu''_i)$ for all $\nu'_i \leq \nu''_i$, we can derive $\nu_i \leq S_i$ for all $i \in V$ in any iteration of Algorithm 2. Since in addition any further iteration implies the increase in at least one weight ν_i, from $\widetilde{\mathcal{S}}_T(W,\alpha,t_\alpha) \neq \emptyset$ the termination of Algorithm 2 follows with $Q = \emptyset$ and $\nu := (\nu_i)_{i\in V} \leq S$ after a finite number of iterations. Finally, with $\nu \in \widetilde{\mathcal{S}}_T(W,\alpha,t_\alpha)$ we can state that ν is equal to the unique minimal point of $\widetilde{\mathcal{S}}_T(W,\alpha,t_\alpha)$. If otherwise $\widetilde{\mathcal{S}}_T(W,\alpha,t_\alpha) = \emptyset$, the algorithm terminates either because of any condition in row 1 or with $Q \neq \emptyset$ after a finite number of iterations.

The time complexity of the algorithm can be deduced as follows. First of all, it can easily be verified that the termination conditions in row 1 and the initialization step can be done with a time complexity of $\mathcal{O}(|V||E|+\mathcal{B}_\alpha)$. Furthermore, the maximum number of iterations is given by $\mathcal{O}(|V|(1+\mathcal{B}))$, which can be followed by the implication that after at most $|V|$ iterations at least one weight ν_i has to be assigned to a succeeding start time component of W_i, since otherwise network N contains a cycle of positive length. Finally, the time complexity of each iteration remains to consider. Obviously, the maximum number of verified and potentially updated weights in each iteration is given by $|E|$. Since each start time restriction W_i of an activity $i \in V$ can be stored in memory as a list of non-decreasing values, representing the start and end times of all start time components, the update process can be done with a time complexity of $\mathcal{O}(|V|+\mathcal{B})$ over all iterations. $\square$

The second temporal planning procedure can be seen as a reversed version of Algorithm 2 which determines the unique maximal point of $\widehat{\mathcal{S}}_T(W, \alpha, t_\alpha) := \{S \in \mathcal{S}_T(W) \mid S_\alpha \leq t_\alpha\}$ or shows $\widehat{\mathcal{S}}_T(W, \alpha, t_\alpha) = \emptyset$ otherwise. Based on an initialized weight with $\nu_\alpha := \max\{\tau \in W_\alpha \mid \tau \leq t_\alpha\}$ and $\nu_i := \infty$ for all other activities, the algorithm decreases the activity weights iteratively until either a W-feasible schedule is determined or $\widehat{\mathcal{S}}_T(W, \alpha, t_\alpha) = \emptyset$ is established. In contrast to the first procedure, the reversed version checks in each iteration for all direct predecessors $i \in \text{Pred}(j)$ of some activity j if the tentative weight ν_i satisfies the minimum time span to ν_j, i.e., $\nu_i \leq \nu_j - \delta_{ij}$.

It is worth mentioning that in Franck et al. (2001a) label-correcting algorithms have already been introduced for a project scheduling problem with calendars which could also be used to determine the unique minimal and maximal point of $\mathcal{S}_T(W)$ with some adjustments. In contrast to the procedures in Franck et al. (2001a), Algorithm 2 and its reversed version provide the possibility to set a lower or upper bound for the start time of any activity of the project without the need to establish further temporal constraints or rather to extend the project network.

4.2 Minimum and maximum time lags

In the following, temporal planning procedures are presented which are essential for some consistency tests as described in Sect. 5. These procedures are able to determine for each activity $i \in V$ all W-feasible start times, where a start time $t \in W_i$ is called W-feasible if at least one schedule $S \in \mathcal{S}_T(W)$ with $S_i = t$ exists. Besides, the procedures can also determine the indirect minimum and maximum time lags between all activity pairs $(i, j) \in V \times V$ of the project for a subset of all time-feasible start times of activity $i \in V$ which can be used to calculate the indirect minimum and maximum time lag for each W-feasible start time as described later on. The indirect minimum (maximum) time lag $\widetilde{d}_{ij}(W, t)$ ($\widehat{d}_{ij}(W, t)$) for any activity pair $(i, j) \in V \times V$ is equal to the time span between the earliest (latest) W-feasible start time of activity j and time t if activity i is assumed to be started not earlier (later) than at time t. That means $t + \widetilde{d}_{ij}(W, t)$ ($t - \widehat{d}_{ij}(W, t)$) corresponds to the earliest (latest) W-feasible start time $\text{ES}_j(W, i, t)$ ($\text{LS}_j(W, i, t)$) of activity $j \in V$ if $\widetilde{\mathcal{S}}_T(W, i, t) \neq \emptyset$ ($\widehat{\mathcal{S}}_T(W, i, t) \neq \emptyset$). It should be noted that in contrast to the temporal planning without start time restrictions, indirect minimum and maximum time lags have to be considered separately due to the start time breaks.

Algorithm 3: Time planning procedure for minimum time lags

```
Input: Start time restriction $W$
Output: Minimum distance matrix $\widetilde{D}(W)$
if $\mathcal{S}_T = \emptyset \vee \exists i \in V : W_i = \emptyset$ then
    terminate ($\mathcal{S}_T(W) = \emptyset$)
forall $i \in V$ do
    $t := \min W_i$
    while $\widetilde{\mathcal{S}}_T(W, i, t) \neq \emptyset$ do
        $ES' := ES(W, i, t)$
        if $ES'_i > t$ then
            $W_i := W_i \setminus [t, ES'_i - 1]$
            $t := ES'_i$
        forall $j \in V \setminus \{i\}$ do
            $d_{ijt} := ES'_j - ES'_i$
        $t' := \min\{\min_{j \in V \setminus \{i\}}\{e(ES'_j) - d_{ij}\}, -d_{i0}\}$
        if $t' > t$ then
            forall $j \in V \setminus \{i\}$ do
                if $ES'_j - d_{ij} < t'$ then
                    if $t < ES'_j - d_{ij}$ then
                        $d_{ij, ES'_j - d_{ij}} := d_{ij}$
                    $d_{ijt'} := d_{ij}$
                else
                    $d_{ijt'} := ES'_j - t'$
        $t := t' + 1$
    $W_i := W_i \setminus [t, \infty)$
return $\widetilde{D}(W) = ([d_{ijt}])_{i,j \in V}$
```

Algorithm 3 determines for each activity $i \in V$ all W-feasible start times and the indirect minimum time lag $\widetilde{d}_{ij}(W, t)$ to any other activity $j \in V \setminus \{i\}$ for each start time $t \in W'_i$ with W'_i as a subset of all time-feasible start times of activity $i \in V$. The indirect minimum time lags $\widetilde{d}_{ij}(W, t)$ for all start times $t \in W'_i$ are stored in a list $[\widetilde{d}_{ij}(W, t)]$ sorted by increasing values of t, where $\widetilde{D}(W) := ([\widetilde{d}_{ij}(W, t)])_{i,j \in V}$ is called the minimum distance matrix of start time restriction W. Algorithm 3 is based on a right-shift over all start times in W_i of an activity $i \in V$, starting with $t := \min W_i$. As long as $\widetilde{\mathcal{S}}_T(W, i, t) \neq \emptyset$, the minimal point $\mathrm{ES}' := \mathrm{ES}(W, i, t)$ is determined. From $\mathrm{ES}'_i > t$, it follows directly that all start times in $\{t, t+1, \dots, \mathrm{ES}'_i - 1\}$ are not W-feasible so that they are removed from W_i in row 8. After that, variable t is set to ES'_i and the indirect minimum time lag between activity i with start time t and each activity $j \in V \setminus \{i\}$ is stored in $d_{ijt} := \mathrm{ES}'_j - \mathrm{ES}'_i$. In the next step, which is based on Theorem 3, a start time t' is calculated to which the currently start time t of activity $i \in V$ could be right-shifted so that all start times $\tau \in [t, t'] \cap W_i$ can be shown to be W-feasible. From Theorem 3, it can easily be derived that for all start times $\tau', \tau'' \in [t, t'] \cap W_i$ with $\tau' < \tau''$ and $\tau'' + d_{ij} \leq \mathrm{ES}'_j$, where d_{ij} corresponds to a longest directed path in network N from activity i to activity j, $\widetilde{d}_{ij}(W, \tau') = \widetilde{d}_{ij}(W, \tau'') + (\tau'' - \tau')$ and that for all other start times with $\tau' + d_{ij} \geq \mathrm{ES}'_j$ the condition $\widetilde{d}_{ij}(W, \tau') = \widetilde{d}_{ij}(W, \tau'')$ is satisfied.

This implies that the storage of $\widetilde{d}_{ij}(W, \mathrm{ES}'_j - d_{ij})$ if $\mathrm{ES}'_j - d_{ij} < t'$ and $\widetilde{d}_{ij}(W, t')$ is sufficient to be able to determine $\widetilde{d}_{ij}(W, \tau)$ for any start time $\tau \in [t, t'] \cap W_i$. The storage of the corresponding values in $[\widetilde{d}_{ij}(W, t)]$ can be seen in rows 13–20 of Algorithm 3. Given list $[\widetilde{d}_{ij}(W, t)]$, the indirect minimum time lag $\widetilde{d}_{ij}(W, \tau)$ of any W-feasible start time τ in interval $[t, t']$ can be calculated by

$$\widetilde{d}_{ij}(W, \tau) := \widetilde{d}_{ij}(W, \tau') + \operatorname{sgn}\big(\widetilde{d}_{ij}(W, \tau'') - \widetilde{d}_{ij}(W, \tau')\big) \cdot (\tau - \tau'),$$

with $\tau' := \max\{\sigma \in \Psi \mid \sigma \leq \tau\}$, $\tau'' := \min\{\sigma \in \Psi \mid \sigma \geq \tau\}$ and Ψ as the set of all start times stored in $[\widetilde{d}_{ij}(W, t)]$. In case that there is still a W-feasible start time $\tau \geq t' + 1$ in W_i, a further loop pass for activity $i \in V$ is conducted. Otherwise, all remaining start times $\tau \geq t' + 1$ in W_i are removed and the next activity is considered. At the end of Algorithm 3, start time restriction W contains all W-feasible start times of the initial start time restriction and the minimum distance matrix $\widetilde{D}(W)$ is returned.

Theorem 3 *Let* $\mathrm{ES}' = \min \widetilde{\mathcal{S}}_\mathrm{T}(W, i, t)$ *be given. Then, for* $\mathrm{ES}'' = \min \widetilde{\mathcal{S}}_\mathrm{T}(W, i, \tau)$ *and each time* $\tau \in W_i$, *satisfying* $\mathrm{ES}'_i \leq \tau \leq \min\{\min_{j \in V \setminus \{i\}}\{e(\mathrm{ES}'_j) - d_{ij}\}, -d_{i0}\}$ *with* $e(\mathrm{ES}'_j)$ *as the end time of the corresponding start time component of* ES'_j, $\mathrm{ES}''_h = \max\{\mathrm{ES}'_h, \tau + d_{ih}\}$ *holds for all* $h \in V$.

Proof Let $\nu = (\nu_h)_{h \in V}$ with $\nu_h = \max\{\mathrm{ES}'_h, \tau + d_{ih}\}$ be given. First of all, it can be derived from $\widetilde{\mathcal{S}}_\mathrm{T}(W, i, t) \supseteq \widetilde{\mathcal{S}}_\mathrm{T}(W, i, \tau)$ that $\mathrm{ES}' \leq \mathrm{ES}''$ ($\mathrm{ES}'_h \leq \mathrm{ES}''_h$ for all $h \in V$). Furthermore, since d_{ih} represents a lower bound for the time span $\mathrm{ES}''_j - \tau$, $(\tau + d_{ih})_{h \in V} \leq \mathrm{ES}''$ follows as well. In conclusion, we get $\nu \leq \mathrm{ES}''$ so that due to $\nu_i = \max\{\mathrm{ES}'_i, \tau + d_{ii}\} = \tau$ ($d_{ii} = 0$) it is sufficient to show that ν is W-feasible to prove $\nu = \mathrm{ES}''$.

First, we show that ν is time-feasible. For this, $\nu_0 = 0$ can be derived by $\mathrm{ES}'_0 = 0$ and condition $\tau \leq -d_{i0}$, equivalent to $\tau + d_{i0} \leq 0$, so that at least one ordered pair $(u, v) \in V \times V$ with $\nu_v < \nu_u + d_{uv}$ exists if ν is not time-feasible. Since ES' is time-feasible, $\mathrm{ES}'_v \geq \mathrm{ES}'_u + d_{uv}$ holds, so that $\nu_u = \tau + d_{iu} > \mathrm{ES}'_u$ can be followed. Finally, we get $\nu_v < \tau + d_{iu} + d_{uv} \leq \tau + d_{iv}$ which contradicts $\nu_v = \max\{\mathrm{ES}'_v, \tau + d_{iv}\}$. To complete the proof, the conditions $\tau \in W_i$ and $\tau \leq \min_{j \in V \setminus \{i\}}\{e(\mathrm{ES}'_j) - d_{ij}\}$ imply $\nu_h \in W_h$ for all $h \in V$, so that ν is W-feasible. □

The maximum distance matrix $\widehat{D}(W) := ([\widehat{d}_{ij}(W, t)])_{i,j \in V}$ which contains the lists $[\widehat{d}_{ij}(W, t)]$ for all activity pairs $(i, j) \in V \times V$ can be calculated in a similar way as described before. The corresponding procedure which also determines all W-feasible start times can be seen as a reversed version of Algorithm 3 which is based on left-shifts over all start times in W_i, considering the latest schedule $\mathrm{LS}(W, i, t)$ in each iteration.

It should be mentioned that in Kreter (2016, Sect. 5.1) and Kreter et al. (2016) different temporal planning procedures have been developed for a project scheduling problem with calendars which could also be used to determine all W-feasible start times of any start time restriction W and the start-time-dependent indirect minimum time lags as described for all activity pairs of the project as well. While for these procedures referred to the problem of this work, time complexities of $\mathcal{O}(\max(|V|^3\bar{d}^3, |V|^4\bar{d}^2))$, $\mathcal{O}(|V|^4\bar{d}^2)$ and $\mathcal{O}(\max(|V|^7(\mathcal{B}+1)^3, |V|^8(\mathcal{B}+1)^2))$ have been shown, Algorithm 3 and its reversed version can both be implemented with a time complexity of $\mathcal{O}(|V|^2|E|(\mathcal{B}+1))$. Furthermore, it should be noted that the procedures from literature are not able to determine the start-time-dependent indirect maximum time lags between all activity pairs of a project.

5 Consistency tests

In the literature, it could already be shown that consistency tests can successively be applied on project scheduling problems with renewable resources in the framework of an exact solution procedure (Dorndorf et al. 2000b; Schutt et al. 2013). Furthermore, in Alvarez-Valdes et al. (2006, 2008) consistency tests have been used in approximation procedures for the RCPSP/π.

Commonly, consistency tests can be seen as pairs of a condition and a constraint, where the constraint is established if the condition is satisfied. In the following, we present five consistency tests whose possibly deduced constraint is unary, i.e., the established constraint can directly be transformed to a reduction in a start time restriction W_i as the domain of start time S_i of activity $i \in V$. Following the terminology in Dorndorf et al. (2000a), such tests can be referred to as domain-consistency tests and can be considered as functions γ mapping a start time restriction W to another start time restriction $W' = \gamma(W)$ with $W'_i \subseteq W_i$ for all $i \in V$. In the following, we call the outcome of all consistency tests from a set Γ, iteratively applied until no domain reduction can be done anymore or $W'_i = \emptyset$ for at least one activity $i \in V$ is detected, a fixed point.

The first two consistency tests are based on the temporal constraints $S_j \geq S_i + \delta_{ij}$ for all $(i, j) \in E$ of problem (P1) and could thus be used for similar project scheduling problems independent of the considered type of resource. At first, we consider a well-known consistency test which has already been used for precedence constraints in Alvarez-Valdes et al. (2006, 2008) and also for general temporal constraints in Dorndorf et al. (2000b). This test is based on the fact that for any temporal constraint $\min W_i + \delta_{ij}$ represents a lower bound of start time S_j and $\max W_j - \delta_{ij}$ gives an upper bound of start time S_i. In this work, the test is called temporal-bound consistency test which is given by the following conditions to be checked for all $(i, j) \in E$ and the corresponding reduction rules for the start time restrictions:

$$\begin{aligned}\min W_j &< \min W_i + \delta_{ij} &&\Rightarrow W_j := W_j \setminus [0, \min W_i + \delta_{ij}[\\ \max W_i &> \max W_j - \delta_{ij} &&\Rightarrow W_i := W_i \setminus \,]\max W_j - \delta_{ij}, \infty[.\end{aligned}$$

It should be noted that the fixed point of the temporal-bound consistency test W' can be obtained with Algorithm 2 and its reversed version with a time complexity of $\mathcal{O}(|V||E|(\mathcal{B}+1))$ by setting $W'_i := W_i \cap [\mathrm{ES}_i(W, 0, 0), \mathrm{LS}_i(W, 0, 0)]$ for all activities $i \in V$.

The second consistency test is based on the temporal constraints as well. In contrast to the first consistency test, all start times $t \in W_i$ of an activity $i \in V$ are checked whether any W-feasible schedule $S \in \mathcal{S}_T(W)$ with $t = S_i$ exists rather than considering only the minimum and maximum start times in W_i. The second test is called temporal consistency test and can be described by the following condition and its corresponding reduction rule:

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

This test is conducted for all activities $i \in V$ and their corresponding start times $t \in W_i$. The fixed point of the temporal consistency test only contains the W-feasible start times for all activities of the project. As described in Sect. 4, the fixed point of the temporal consistency test can be determined by Algorithm 3 with a time complexity of $\mathcal{O}(|V|^2|E|(\mathcal{B}+1))$. It is easy to verify that the temporal consistency test dominates the temporal-bound consistency test, which means that $W_i^b \supseteq W_i^t$ holds for all $i \in V$ with W^b and W^t as the fixed points of the temporal-bound and temporal consistency test, respectively.

The following consistency tests take the resource constraints into consideration. The first of these consistency tests is used to remove each start time from the start time restriction of some activity if this start time implies a resource conflict taking the minimum possible resource consumptions of all other activities into account. Accordingly, for this test the minimum resource consumption of each resource $k \in \mathcal{R}$ by an activity $i \in V_k$ is determined if it is assumed that the activity can be started at any time in W_i. That means $r_{ik}^{c,\min}(W) := \min\{r_{ik}^c(\tau) \mid \tau \in W_i\}$ is calculated for each $i \in V$ and $k \in \mathcal{R}_i := \{k \in \mathcal{R} \mid r_{ik}^d > 0\}$, where the so-called resource-bound consistency test for each start time $t \in W_i$ of any activity $i \in V \setminus \{0, n+1\}$ is stated by

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

It should be noted that one pass over all activities and start times can be conducted with a time complexity of $\mathcal{O}(|V|\mathcal{I} + |\mathcal{R}|\mathcal{B})$ with $\mathcal{I}$ as the number of components over all sets Π_k, where $I := \{a, a+1, \ldots, b\} \subset \mathbb{Z}$ is called a component of Π_k if $I \subseteq \Pi_k$ and $a-1, b+1 \notin \Pi_k$ is given. This can be achieved by storing the resource usages $r_{ik}^u(t)$ for each activity $i \in V$ and resource $k \in \mathcal{R}_i$ for a subset of the start times of the whole planning horizon H in a list $[r_{ik}^u(t)]$ sorted by increasing values of t which is sufficient to calculate the resource usage for any start time just like for the lists $[\widetilde{d}_{ij}(W,t)]$ and $[\widehat{d}_{ij}(W,t)]$. The resource usages of the start times $\tau \in H$ which have to be stored in $[r_{ik}^u(t)]$ can be deduced from the following relations:

$$\begin{aligned}
\tau + 1 \in \Pi_k \wedge \tau + p_i + 1 \in \Pi_k &\Rightarrow r_{ik}^u(\tau+1) = r_{ik}^u(\tau)\\
\tau + 1 \in \Pi_k \wedge \tau + p_i + 1 \notin \Pi_k &\Rightarrow r_{ik}^u(\tau+1) = r_{ik}^u(\tau) - 1\\
\tau + 1 \notin \Pi_k \wedge \tau + p_i + 1 \in \Pi_k &\Rightarrow r_{ik}^u(\tau+1) = r_{ik}^u(\tau) + 1\\
\tau + 1 \notin \Pi_k \wedge \tau + p_i + 1 \notin \Pi_k &\Rightarrow r_{ik}^u(\tau+1) = r_{ik}^u(\tau).
\end{aligned}$$

The given relations imply that it is sufficient to store the resource usages $r_{ik}^u(\tau)$ of a resource $k \in \mathcal{R}$ by an activity $i \in V$ for all start times $\tau \in \mathcal{U}_k$ or $\tau + p_i \in \mathcal{U}_k$ with

$\mathcal{U}_k := \{\sigma \mid \sigma \in \Pi_k \wedge \sigma + 1 \notin \Pi_k\} \cup \{\sigma \mid \sigma \notin \Pi_k \wedge \sigma + 1 \in \Pi_k\} \cup \{0, \bar{d}\}$, so that the resource usage for any start time $\tau \in H$ can be calculated by

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

with $\tau' := \max\{\sigma \in \Psi \mid \sigma \leq \tau\}$, $\tau'' := \min\{\sigma \in \Psi \mid \sigma \geq \tau\}$ and Ψ as the set of all start times stored in $[r^u_{ik}(t)]$. It follows directly that the maximum number of start times stored in any list $[r^u_{ik}(t)]$ is polynomially bounded by $\mathcal{O}(\mathcal{I}_k)$ with $\mathcal{I}_k$ as the number of components in Π_k, so that it can easily be verified that $r^{c,\min}_{ik}(W)$ can be determined with a time complexity of $\mathcal{O}(\mathcal{I}_k + \mathcal{B}_i)$, which results in a time complexity of $\mathcal{O}(|V|\mathcal{I} + |\mathcal{R}|\mathcal{B})$ over all activities $i \in V$ and all resources $k \in \mathcal{R}$. Since all inconsistent start times $t \in W_i$ with respect to the resource-bound consistency test for any resource $k \in \mathcal{R}$ can be removed from W_i with a time complexity of $\mathcal{O}(\mathcal{I}_k + \mathcal{B}_i)$, in conclusion we get a time complexity of $\mathcal{O}(|V|\mathcal{I} + |\mathcal{R}|\mathcal{B})$ for one pass of the resource-bound consistency test over all activities and resources.

The next consistency test can be seen as an extension of the resource-bound consistency test, where besides the resource constraints the temporal constraints between the activities of the project are considered as well. Thereby, this test makes use of the fact that each activity of the project has to be started in a schedule-dependent time window if the start time of some activity is fixed, so that the calculation of the minimum resource consumptions can be restricted to these time windows. For the so-called D-interval consistency test, the distance matrix $D = (d_{ij})_{i,j \in V}$ or rather the lengths of the longest directed paths d_{ij} in network N between all activity pairs $(i, j) \in V \times V$ are used to restrict the possible start times $\tau \in W_j$ of each activity $j \in V \setminus \{i\}$ to start times in $[t + d_{ij}, t - d_{ji}]$ with t as the given start time of activity $i \in V$. The consistency test is conducted for each D-consistent start time $t \in W_i$ of an activity $i \in V$, where a start time $t \in W_i$ is called D-consistent if and only if $[t + d_{ij}, t - d_{ji}] \cap W_j \neq \emptyset$ for all $j \in V$. Considering any D-consistent start time of activity $i \in V$, $r^c_{ik}(t)$ and the minimum resource consumption of each activity $j \in V \setminus \{i\}$ over all start times $\tau \in [t + d_{ij}, t - d_{ji}] \cap W_j$ is determined, i.e., $r^{c,\min}_{ijkt}(W, D) := \min\{r^c_{jk}(\tau) \mid \tau \in W_j \cap [t + d_{ij}, t - d_{ji}]\}$ is calculated. Conclusively, the D-interval consistency test can be stated by

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

for each activity $i \in V$ and each D-consistent start time $t \in W_i$.

One pass of the D-interval consistency test over all D-consistent start times of all activities $i \in V$ is outlined in Algorithm 4. As it can be seen in rows 4 and 5, the algorithm is based on the generation of lists which can be used in the same manner as the lists described before.

Algorithm 4: D-interval consistency test

```
Input: Start time restriction $W$
Output: $\gamma^D(W)$

1 forall $i \in V$ do
2     forall $k \in \mathcal{R}$ do
3         forall $j \in V_k \setminus \{i\}$ do
4             Determine list $[r^{u,min}_{ijkt}(W, D)]$
5         Compute list $[r^{c,min}_{ikt}(W, D)]$
6 Update $W$ $(\exists k \in \mathcal{R} : r^{c,min}_{ikt} > R_k \;\Rightarrow\; W_i := W_i \setminus \{t\})$
7 return $W$
```

First of all, we consider the generation of list $[r^{u,\min}_{ijkt}(W, D)]$, which contains the minimum resource usages

$$r^{u,\min}_{ijkt}(W, D) := \min\{r^u_{jk}(\tau) \mid \tau \in W_j \cap [t + d_{ij}, t - d_{ji}]\}$$

of all start times in a subset of all D-consistent start times of activity $i \in V$. The generation of the list is based on a right-shift over all start times $t \in W_i$ of activity $i \in V$. Let $t \in W_i$ be any D-consistent start time of activity $i \in V$. Then, the minimum resource usage of activity $j \in V \setminus \{i\}$ in the restricted start time set $W^r_j := [t + d_{ij}, t - d_{ji}] \cap W_j \neq \emptyset$ is given by $r^u_{\min} := \{r^u_{jk}(\tau) \mid \tau \in W^r_j\}$, where $\tau^{\min} := \max\{\tau \in W^r_j \mid r^u_{jk}(\tau) = r^u_{\min}\}$ represents the greatest start time in W^r_j with the lowest resource usage. The greatest start time t^s to which the current start time t of activity $i \in V$ can be right-shifted so that $r^u_{\min}$ keeps unchanged, is based on the calculation of $\tau' := \min\{\tau \in W_j \mid \tau > t - d_{ji} \wedge r^u_{jk}(\tau) < r^u_{\min}\}$ and $\tau'' := \min\{\tau \in W_j \mid \tau > \tau^{\min} \wedge r^u_{jk}(\tau) > r^u_{\min}\}$ representing the next start times of activity j with a lower or greater resource usage than $r^u_{\min}$. That means the minimum resource usage $r^u_{\min}$ does not decrease for all start times t of activity $i \in V$ with $t - d_{ji} < \tau'$ and does not increase for all start times with $t + d_{ij} \leq \max\{\tau \in W_j \mid \tau < \tau''\}$, so that $t^s := \min\{\tau' - 1 + d_{ji}, \max\{\tau \in W_j \mid \tau < \tau''\} - d_{ij}\}$ can directly be deduced. After the storage of $r^u_{\min}$ in list $[r^{u,\min}_{ijkt}(W, D)]$ for start time t^s, the minimum resource usage $r^u_{\min}$ of the next start time t^+ with $W^r_j \neq \emptyset$ is stored as well. In case that for some directly succeeding start times $t > t^+$ each right-shift by one unit leads to an increase or decrease in $r^u_{\min}$ by exactly one unit, the greatest of these start times and its corresponding minimum resource usage $r^u_{\min}$ is also stored in $[r^{u,\min}_{ijkt}(W, D)]$. If the described procedure is reiterated until all start times in W_i are considered, $[r^{u,\min}_{ijkt}(W, D)]$ is determined, which is implemented with a time complexity of $\mathcal{O}(\mathcal{I}^2_k + \mathcal{B}^2_j)$.

After the generation of the lists $[r^{u,\min}_{ijkt}(W, D)]$ for all $j \in V_k \setminus \{i\}$, a list for the minimum resource consumption

$$r^{c,\min}_{ikt}(W, D) := r^c_{ik}(t) + \sum_{j \in V_k \setminus \{i\}} r^{c,\min}_{ijkt}(W, D)$$

can trivially be deduced by calculating the resource consumptions for all start times stored in the lists $[r^{u,\min}_{ijkt}(W, D)]$ and $[r^u_{ik}(W, t)]$. Thus, the maximum number of stored start times in $[r_{ikt}^{c,\min}(W, D)]$ is given by $\mathcal{O}(|V|\mathcal{I}_k^2 + \mathcal{B}^2)$, which is equal to the time complexity of the update procedure for each resource $k \in \mathcal{R}$ at the end of the algorithm. In conclusion, Algorithm 4 returns a start time restriction $\gamma^D(W)$ with a time complexity of $\mathcal{O}(|V|^2\mathcal{I}^2 + |V||\mathcal{R}|\mathcal{B}^2)$.

The last consistency test extends the D-interval consistency test in the sense that for each W-feasible start time $t \in W_i$ of an activity $i \in V$ the set of start times considered for each activity $j \in V_k \setminus \{i\}$ is restricted even more by using the minimum and maximum distance matrices $\widetilde{D}(W)$ and $\widehat{D}(W)$. That means for any W-feasible start time $t \in W_i$ of activity $i \in V$ the considered start times $\tau \in W_j$ of activity $j \in V_k \setminus \{i\}$ are restricted to $[t + \widetilde{d}_{ij}(W, t), t - \widehat{d}_{ij}(W, t)]$. Accordingly, the minimum consumption of resource $k \in \mathcal{R}$ by activity $j \in V_k \setminus \{i\}$, which is dependent on the W-feasible start time $t \in W_i$ of activity $i \in V$, is given by

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

The corresponding condition and reduction rule of the so-called W-interval consistency test for all activities $i \in V$ and all W-feasible start times $t \in W_i$ are given by

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

One pass over all W-feasible start times of all activities $i \in V$ can be conducted in the same way as for the D-interval consistency test sketched in Algorithm 4. In contrast to the D-interval consistency test, a list $[r_{ijkt}^{u,\min}(W, \widetilde{D}, \widehat{D})]$ is determined for each activity $j \in V_k \setminus \{i\}$ with

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

The generation of this list works quite similar to Algorithm 4, except that for the right-shifts over the start times of activity $i \in V$, intervals with constant courses of $\widetilde{d}_{ij}(W, t)$ and $\widehat{d}_{ij}(W, t)$ have to be taken into consideration as well. Since the maximum number of start times stored in lists $[\widetilde{d}_{ij}(W, t)]$ and $[\widehat{d}_{ij}(W, t)]$ is given by $\mathcal{O}(\mathcal{B} + 1)$, the time complexity for the generation of $[r_{ijkt}^{u,\min}(W, \widetilde{D}, \widehat{D})]$ equals $\mathcal{O}((\mathcal{I}_k + \mathcal{B}_j)(\mathcal{I}_k + \mathcal{B}_j + \mathcal{B}))$. Just like for the D-interval consistency test, the list $[r_{ikt}^{c,\min}(W, \widetilde{D}, \widehat{D})]$ with

$$r_{ikt}^{c,\min}(W, \widetilde{D}, \widehat{D}) := r_{ik}^c(t) + \sum_{j \in V_k \setminus \{i\}} r_{ijkt}^{c,\min}(W, \widetilde{D}, \widehat{D})$$

for each resource $k \in \mathcal{R}$ can trivially be deduced and the update of W_i follows directly. Conclusively, the W-interval consistency test determines $\gamma^W(W)$ with a time complexity of $\mathcal{O}(|V|^2\mathcal{I}^2 + |V|^2\mathcal{I}\mathcal{B} + |V||\mathcal{R}|\mathcal{B}^2)$.

6 Lower bounds

In the following, we describe two lower bounds for the project duration which can be used for any node in the search tree or rather its corresponding start time restriction W. The first lower bound LB0^π is given by the solution of problem (P2(W)), i.e., $\text{LB0}^\pi := \text{ES}_{n+1}(W)$ with $\text{ES}(W) := \text{ES}(W, 0, 0)$. In the literature, such a lower bound based on a relaxation is usually referred as constructive. Conversely, the second lower bound LBD^π is termed destructive, meaning that a hypothetical maximum project duration d is increased as long as it can be shown that d precludes any feasible solution (Klein and Scholl 1999). Algorithm 5 shows the procedure to determine the destructive lower bound LBD^π , where the structure of Algorithm 5 is inspired by Franck et al. (2001b) and the way to find the greatest hypothetical project duration, which cannot be rejected anymore, is taken from Klein and Scholl (1999).

Algorithm 5: Destructive lower bound LBD^π

```
Input: Start time restriction $W$, $LB^{start}$, $UB^{start}$
Output: $LBD^\pi$
1  $LB^d := LB^{start}$   $UB^d := UB^{start}$
2  $ES' := \min \widetilde{\mathcal{S}}_T(W, n+1, LB^{start})$
3  while $LB^d \leq UB^d$ do
4      $d := \lceil (LB^d + UB^d)/2 \rceil$
5      $LS' := \max \widehat{\mathcal{S}}_T(W, n+1, d)$
6      if $\exists k \in \mathcal{R} : \sum_{i \in V_k} r_{ik}^{c,min}(W, ES'_i, LS'_i) > R_k$ then
7          if $d = UB^{start}$ then
8              terminate
9          $LB^d := d + 1$
10     else
11         $UB^d := d - 1$
12 $LBD^\pi := LB^d$
13 return $LBD^\pi$
```

Algorithm 5 determines for any enumeration node or rather its corresponding start time restriction W the lowest hypothetical maximum project duration (if it exists) on a given interval $[\text{LB}^{\text{start}}, \text{UB}^{\text{start}}]$ which does not contradict the existence of a feasible schedule, where the verification of the existence of a feasible schedule is described later on. First of all, the algorithm starts with $\text{LB}^{\text{start}} := \max(\text{LB0}^\pi, \text{LB}^{\text{G}})$, where LB^{G} represents the global lower bound determined in the root node, and $\text{UB}^{\text{start}} := \min(\bar{d}, S^*_{n+1} - 1)$ with S^* as the currently best found solution (if already detected). In the following, it is assumed that $\text{LB}^{\text{start}} \leq \text{UB}^{\text{start}}$ and $\widetilde{\mathcal{S}}_{\text{T}}(W, n+1, \text{LB}^{\text{start}}) \neq \emptyset$ is given, since otherwise, we can state that LBD^π is greater than UB^{start} so that the algorithm does not have to be conducted. The main step of Algorithm 5 executes a binary search on interval $[\text{LB}^{\text{start}}, \text{UB}^{\text{start}}]$ in the following way. In each iteration, an interval $[\text{LB}^d, \text{UB}^d]$ is considered with $\text{LB}^d := \text{LB}^{\text{start}}$ and $\text{UB}^d := \text{UB}^{\text{start}}$ for the first iteration. Based on this interval, $d := \lceil (\text{LB}^d + \text{UB}^d)/2 \rceil$ is determined. If it can be shown that the maximum project duration d precludes the existence of any feasible solution, it can be followed that LBD^π has to be greater than d. In the case that $d = \text{UB}^{\text{start}}$, the algorithm terminates since LBD^π is known to be greater than UB^{start}.

Otherwise, for the next iteration $\mathrm{LB}^d := d + 1$ is set, so that the interval $[d + 1, \mathrm{UB}^d]$ is considered next. Conversely, if d cannot be rejected, the interval $[\mathrm{LB}^d, d - 1]$ is investigated in the next iteration, i.e., $\mathrm{UB}^d := d - 1$ is set. This procedure is reiterated while $\mathrm{LB}^d \leq \mathrm{UB}^d$ is given, where LB^d equals the destructive lower bound LBD^π at the end of the algorithm.

Finally, the way the algorithm verifies if any feasible solution exists for a given maximum project duration d on interval $[\mathrm{LB}^{\mathrm{start}}, \mathrm{UB}^{\mathrm{start}}]$ remains to consider. Let d be any hypothetical maximum project duration on interval $[\mathrm{LB}^{\mathrm{start}}, \mathrm{UB}^{\mathrm{start}}]$. Then, the minimum resource consumption for each resource $k \in \mathcal{R}$ by an activity $i \in V_k$ over all start times $t \in W_i \cap [\mathrm{ES}'_i, \mathrm{LS}'_i]$ is determined with ES'_i as the earliest and LS'_i as the latest W-feasible start time of activity $i \in V_k$ if the project duration is not lower than $\mathrm{LB}^{\mathrm{start}}$ and not greater than d. It should be noted that in contrast to ES'_i, LS'_i has to be determined in each iteration since it depends on d. Given $r_{ik}^{c,\min}(W, t_1, t_2) := \min\{r_{ik}^c(\tau) \mid \tau \in W_i \cap [t_1, t_2]\}$, the minimum resource consumption over all start times $t \in W_i \cap [\mathrm{ES}'_i, \mathrm{LS}'_i]$ can be expressed by $r_{ik}^{c,\min}(W, \mathrm{ES}'_i, \mathrm{LS}'_i)$ and thus the total minimum resource consumption of a resource $k \in \mathcal{R}$ by $\sum_{i \in V_k} r_{ik}^{c,\min}(W, \mathrm{ES}'_i, \mathrm{LS}'_i)$. If there exists at least one resource $k \in \mathcal{R}$ with a total minimum resource consumption greater than R_k, the considered maximum project duration d precludes any feasible solution. Otherwise, the existence of a feasible solution with project duration d cannot be ruled out by the described procedure.

In the following, the time complexities of both lower bounds are considered. For the first lower bound $\mathrm{LB0}^\pi$, a time complexity of $\mathcal{O}(|V||E|(\mathcal{B}+1))$ has already been shown in Sect. 4, where it should be noted that the determination of $\mathrm{LB0}^\pi$ is already a part of the enumeration process itself, so that it does not cause any additional computational effort. In contrast, the destructive lower bound LBD^π entails additional computing time with a possibly better lower bound as an outcome, i.e., $\mathrm{LBD}^\pi \geq \mathrm{LB0}^\pi$. For Algorithm 5, we can state a time complexity of $\mathcal{O}(\log(\bar{d})(|V||E|(\mathcal{B}+1)+|\mathcal{R}|\mathcal{B}+|V|\mathcal{I}))$ based on the following observations. First of all, the maximum number of iterations is given by $\log(\bar{d})$ due to the binary search. Combined with the time complexity to determine LS' given by $\mathcal{O}(|V||E|(\mathcal{B} + 1))$, and the time complexity to get the total minimum resource consumption of all resources stated by $\mathcal{O}(|\mathcal{R}|\mathcal{B} + |V|\mathcal{I})$, the mentioned time complexity follows.

7 Dominance rules

In the following, two dominance rules are described where both have in common that they implicate for two nodes, given by their corresponding start time restrictions W' and W'', that $\mathcal{S}(W') \subseteq \mathcal{S}(W'')$ with $\mathcal{S}(W) := \mathcal{S}_\mathrm{T}(W) \cap \mathcal{S}$ holds. Obviously, if both nodes are not reachable from each other in the search tree, this implicates the redundancy of the enumeration node with start time restriction W' or rather the dominance of the enumeration node with start time restriction W''.

For the first dominance rule, a so-called resource usage bound $\bar{U} := (\bar{u}_{ik})_{i \in V, k \in \mathcal{R}}$ is assigned to each enumeration node in the search tree, used to store all resource usage bounds $\bar{u}_{ik}$ established during the enumeration process as described in Sect. 3. Since at the beginning of the enumeration process no resource usage restriction is considered, $\bar{u}_{ik} := p_i$ for all $i \in V$ and $k \in \mathcal{R}$ is set for the root node. In order to represent all time-feasible start times of an activity for a node in the enumeration tree, satisfying all resource usage bounds $\bar{u}_{ik}$ established during the enumeration process, we introduce further notations. First, we define the so-called $\bar{U}$-induced start time restriction of an activity $i \in V$ by $W_i(\bar{U}) := \bigcap_{k \in \mathcal{R}} W_{ik}(\bar{u}_{ik})$ and call $W(\bar{U}) := (W_i(\bar{U}))_{i \in V}$ the corresponding $\bar{U}$-induced start time restriction. For the following explanations, in order to improve the readability, we write $W' \subseteq W''$ instead of $W_i' \subseteq W_i''$ for all $i \in V$ and $\bar{U}' \leq \bar{U}''$ to state $\bar{u}_{ik}' \leq \bar{u}_{ik}''$ for all $i \in V$ and $k \in \mathcal{R}$.

The first rule, called $\bar{U}$-dominance rule, compares the resource usage bounds of nodes which are not reachable from each other in the search tree to reveal redundancies. Let $\bar{U}'$ and $\bar{U}''$ be the resource usage bounds of such nodes and assume that $\bar{U}' \leq \bar{U}''$ is given. Then, the $\bar{U}$-dominance rule detects that the node corresponding to $\bar{U}'$ is redundant or rather dominated by the other node which can be deduced as follows. First of all, let W be the start time restriction and $\bar{U}$ the resource usage bound of an arbitrary node in the search tree. Then it can easily be verified that $W \subseteq W(\bar{U})$ and $\mathcal{S}(W) = \mathcal{S}(W(\bar{U}))$ is given, since no consistency test excludes feasible schedules from $\mathcal{S}_\mathrm{T}(W)$ (cf. Sect. 5). Since $\bar{U}' \leq \bar{U}''$ implicates $\mathcal{S}_\mathrm{T}(W(\bar{U}')) \subseteq \mathcal{S}_\mathrm{T}(W(\bar{U}''))$ and thus $\mathcal{S}(W(\bar{U}')) \subseteq \mathcal{S}(W(\bar{U}''))$ as well, $\mathcal{S}(W') \subseteq \mathcal{S}(W'')$ follows directly with W' and W'' as the start time restrictions corresponding to the nodes with $\bar{U}'$ and $\bar{U}''$, respectively.

In contrast to the first rule, the second rule is not dependent on storing additional information for each enumeration node. Instead, it directly compares the start time restrictions W' and W'' of nodes which are not reachable from each other in the search tree, for what reason the rule is called W-dominance rule. The redundancy of a node is detected by the rule if the condition $W' \subseteq W''$ is satisfied, where the dominance of the node corresponding to W'' is trivially given by $\mathcal{S}_\mathrm{T}(W') \subseteq \mathcal{S}_\mathrm{T}(W'')$.

Concluding, the time complexity for each dominance rule should be considered, where the time complexity refers to the dominance verification between two given nodes. For the $\bar{U}$-dominance rule, a time complexity of $\mathcal{O}(|V||\mathcal{R}|)$ can obviously be determined and for the W-dominance rule a time complexity of $\mathcal{O}(|V| + \min(\mathcal{B}', \mathcal{B}''))$ can be stated with $\mathcal{B}'$ and $\mathcal{B}''$ as the numbers of start time breaks corresponding to W' and W'', respectively.

8 Partitioning the feasible region

The dominance rules described in the previous section are just able under specified conditions to avoid that one and the same part of $\mathcal{S}_\mathrm{T}$ is explored several times in the search tree. In contrast, the following procedure ensures that any part of $\mathcal{S}_\mathrm{T}$ is explored at most one time by partitioning the feasible region of each enumeration node. It should be noted that a similar approach has already been used in Murty (1968) for the assignment problem. In order to achieve the partitioning for each node, the enumeration scheme has to be adjusted as follows. Let W be the start time restriction corresponding to any enumeration node in the search tree with $S = \min \mathcal{S}_\mathrm{T}(W) \notin \mathcal{S}_\mathrm{R}$ and the chosen conflict resource $k \in \mathcal{R}$. Furthermore, assume that $(i_1, i_2, \ldots, i_\mu, \ldots, i_{|V_k(S)|})$

is an arbitrary sequence of all activities considered for the decomposition of $\mathcal{S}_\mathrm{T}(W)$ as described in Sect. 3 with $V_k(S) := \{i \in V_k \mid r^u_{ik}(S_i) > 0\}$. Then, the start time restriction W^{i_μ} corresponding to $i_\mu \in V_k(S)$ for each $\mu \in \{1, 2, \ldots, |V_k(S)|\}$ is set to $W^{i_\mu}_i := W_i$ for all $i \in V \setminus V_k(S)$ and

$$W^{i_\mu}_{i_s} := \begin{cases} W_{i_s}, & \text{if } s < \mu \\ \{\tau \in W_{i_s} \mid r^u_{i_s,k}(\tau) < r^u_{i_s,k}(S)\}, & \text{if } s = \mu \\ \{\tau \in W_{i_s} \mid r^u_{i_s,k}(\tau) \geq r^u_{i_s,k}(S)\}, & \text{otherwise} \end{cases}$$

for all $s \in \{1, 2, \ldots, |V_k(S)|\}$.

In the following, we will show that the described decomposition leads to a partition of $\mathcal{S}_\mathrm{T}(W)$ satisfying $\mathcal{S}_\mathrm{T}(W) \cap \mathcal{S} = \bigcup_{i \in V_k(S)}(\mathcal{S}_\mathrm{T}(W^i) \cap \mathcal{S})$ so that the correctness of the enumeration scheme with the adjusted decomposition still remains. First of all, a partition of $\mathcal{S}_\mathrm{T}(W)$ is given if $\mathcal{S}_\mathrm{T}(W^{i_{\mu'}}) \cap \mathcal{S}_\mathrm{T}(W^{i_{\mu''}}) = \emptyset$ holds for all $\mu', \mu'' \in \{1, 2, \ldots, |V_k(S)|\}$ with $\mu' \neq \mu''$ which follows directly from the guideline for the decomposition. Thus, it remains to show that any feasible schedule in $\mathcal{S}_\mathrm{T}(W)$ is an element of the feasible region of any child node. For this, it is sufficient to show that

$$\bigcup_{\mu=1}^{|V_k(S)|} \mathcal{S}_\mathrm{T}(\overline{W}^{i_\mu}) = \bigcup_{\mu=1}^{|V_k(S)|} \mathcal{S}_\mathrm{T}(W^{i_\mu}) \tag{1}$$

holds with $\overline{W}^{i_\mu}$ as the start time restriction determined in the enumeration procedure for activity $i_\mu \in V_k(S)$ as described in Sect. 3. The correctness of Equation (1) can be deduced from $\mathcal{S}_\mathrm{T}(W^{i_{|V_k(S)|}}) = \mathcal{S}_\mathrm{T}(\overline{W}^{i_{|V_k(S)|}})$, $\mathcal{S}_\mathrm{T}(\overline{W}^{i_\mu}) \setminus \mathcal{S}_\mathrm{T}(W^{i_\mu}) = \bigcup_{\mu'=\mu+1}^{|V_k(S)|} \mathcal{S}_\mathrm{T}(W^{i_{\mu'}})$ for all $\mu = \{1, 2, \ldots, |V_k(S)|-1\}$ and $\mathcal{S}_\mathrm{T}(W^i) \subseteq \mathcal{S}_\mathrm{T}(\overline{W}^i)$ for all $i \in V_k(S)$.

9 Branch-and-bound procedure

In this section, the framework of our branch-and-bound procedure for the RCPSP/max-π is covered, which means that the corresponding representation of the procedure enables different specifications. Besides the enumeration scheme, the branch-and-bound procedure is given by a search strategy which determines how the search tree is built, consistency tests which are applied on start time restrictions, lower bounds on the project duration, and dominance rules used to prune redundant parts of the search tree.

For the following explanations, we assume the search strategy to be subdivided into different strategies, called traversing, generation, ordering, and branching strategy. The traversing strategy determines the sequence in which all not completely explored nodes in the search tree are considered, where a node is said to be completely explored exactly if all its child nodes are generated. As traversing strategy, the well-known depth-first search strategy (DFS) is applied. Since computational tests have shown that DFS results in a long calculation time to find a first feasible solution, leading to a rather bad performance especially for great instance sets, an additional traversing strategy has been implemented to enhance the diversification in the search tree. This strategy works just like the DFS, except that after a predefined time span one of all not completely explored nodes with lowest level in the search tree and lowest lower bound is considered next. We call this traversing strategy scattered-path search (SPS) and denote by SPS$^+$ the extension which considers priority values of the nodes in addition, where the priority values are determined in the same way as for the ordering strategy as it will be described later on.

It should be noted that the traversing strategy neither states the maximum number of child nodes to be generated for any explored node nor the order in which the child nodes are considered. Instead, these specifications for the search procedure are determined by the generation and the ordering strategy, respectively. For the generation strategy, considering any search node which is explored, we distinguish between the alternatives to generate all its child nodes (all) or to restrict the number of the generated child nodes by a maximum value (restr). Besides the maximum number for the generation of child nodes, the order in which they are considered during the search procedure has also been shown to be crucial for the performance by computational studies. As it has already been observed for the RCPSP/max, it is also beneficial for the RCPSP/max-π to explore all generated child nodes in an order of non-decreasing lower bounds which can be seen to increase locally the probability to find a good solution. Since it is likely that the lower bounds among some child nodes are equal, the ordering strategy enables the usage of priority values for the child nodes in addition to identifying the most favorable ones. In the following, the most promising priority values, we are aware of, are presented. For this, let W be the start time restriction of any search node with $S := \min \mathcal{S}_{\mathrm{T}}(W) \notin \mathcal{S}_{\mathrm{R}}$ and assume that $k \in \mathcal{R}^c := \{k \in \mathcal{R} \mid r_k^c(S) > R_k\}$ is the chosen conflict resource for the decomposition of $\mathcal{S}_{\mathrm{T}}(W)$. Furthermore, assume that $(i_1, i_2, \ldots, i_s)$ is a sequence of all generated child nodes sorted by non-decreasing lower bounds on the project duration with $i_\mu \in V_k^c := V_k(S) = \{i \in V \mid r_{ik}^u(S_i) > 0\}$ for all $\mu \in \{1, 2, \ldots, s\}$ and $s \leq |V_k^c|$, so that $i_{\mu'}$ is explored before $i_{\mu''}$ if $\mu' < \mu''$ holds. Then, the sequence between the activities with equal lower bounds in $(i_1, i_2, \ldots, i_s)$ is additionally sorted by priority values dependent on the chosen priority rule, where the corresponding activities can be sorted by either non-decreasing (min) or non-increasing (max) priority values for each rule. For the first two rules, a priority value π_i is assigned to each activity $i \in V_k^c$ based on the resource usage induced by schedule S, with $\pi_i = r_{ik}^u(S_i)$ for the so-called resource usage rule (RU) and $\pi_i = r_{ik}^u(S_i)/p_i$ for the resource-usage-processing-time-ratio rule (RUPT). The delayed-start-time rule (DST) takes the minimum right-shift of the currently start time S_i for each activity $i \in V_k^c$, caused by the resource usage restriction of the enumeration process, into consideration, i.e., $\pi_i = \Delta_{ik}$ with

$$\Delta_{ik} := \begin{cases} \min\{\tau \in W_i(\bar{u}_{ik}) \mid \tau \geq S_i\} - S_i, & \text{if } \{\tau \in W_i(\bar{u}_{ik}) \mid \tau \geq S_i\} \neq \emptyset \\ \max W_i - S_i + 1, & \text{otherwise} \end{cases}$$

and $\bar{u}_{ik} := r_{ik}^u(S_i) - 1$. In contrast to the aforementioned priority rules, the following rules are not dependent on the conflict resource $k \in \mathcal{R}^c$. Instead, those rules are based on float or slack times which are adapted to be able to take start time restrictions into consideration. The first priority rule (TF) determines the so-called total float TF_i for each activity $i \in V_k^c$, which is defined as the maximum right-shift of start time S_i so that in $\mathcal{S}_{\mathrm{T}}(W)$ any W-feasible schedule with a better project duration, than that one of the best feasible solution found so far, exists. Thus, the priority value π_i of each activity $i \in V_k^c$ is given by

$$\mathrm{TF}_i := \mathrm{LS}_i^{\mathrm{UB}}(W) - S_i,$$

with $\mathrm{LS}_i^{\mathrm{UB}}(W) := \mathrm{LS}_i(W, n+1, \mathrm{UB}-1)$ and UB as the project duration of the best feasible solution in case that any solution has already been found or $\mathrm{UB} := \bar{d} + 1$ otherwise. The last priority rule (EFF) is based on the early free float EFF_i of an activity $i \in V_k^c$ which is equal to the maximum possible right-shift of start time S_i so that all other activities of the project can still be started to their earliest W-feasible start time. Hence,

$$\mathrm{EFF}_i := \max\{\tau \in W_i \mid \tau \leq \min_{j \in \mathrm{Succ}(i)} (S_j - \delta_{ij})\} - S_i$$

represents the priority value π_i for each activity $i \in V_k^c$. Conclusively, for the priority rules DST, TF and EFF a further variant has been implemented so that the number of start times skipped in the corresponding start time restriction by the right-shift is considered instead. That means $\pi_i = |]S_i, S_i + \Delta_i] \cap W_i|$ with $\Delta_i \in \{\mathrm{TF}_i, \mathrm{EFF}_i, \Delta_{ik}\}$ is determined for each activity $i \in V_k^c$ for the corresponding rule, where these variants are denoted by $\mathrm{DST}^{\mathrm{I}}$, TF^{I} and $\mathrm{EFF}^{\mathrm{I}}$, respectively. The ordering, based on the priority values as described, can also be used to determine a sequence for the generation of all child nodes of an enumeration node. For this generation strategy, a candidate list of all child nodes is created, sorted by the priority values used for the ordering strategy to determine the sequence to generate the child nodes (restrCL). It should be noted that this strategy is just useful if the number of generated child nodes is restricted.

The last part of the search strategy is given by the branching strategy which determines the way to choose a conflict resource $k \in \mathcal{R}^c$ for any unexplored node to decompose the corresponding feasible region $\mathcal{S}_{\mathrm{T}}(W)$. As the ordering of nodes, the selection of any conflict resource $k \in \mathcal{R}^c$ is related to priority values as well. Based on the priority rules for the ordering strategy, same named priority rules for the branching strategy are directly derived by assigning $\pi_k = \sum_{i \in V_k^c} \pi_i / |V_k^c|$ to each conflict resource $k \in \mathcal{R}^c$. This means, for example, that the branching strategy TF assigns $\pi_k = \sum_{i \in V_k^c} \mathrm{TF}_i / |V_k^c|$ to each conflict resource $k \in \mathcal{R}^c$ which is equal to the average total float over all activities in V_k^c. Besides the priority rules which are related to the priority values of the ordering strategies, additional rules are considered for the branching strategy. These rules assign to the priority value π_k of each conflict resource $k \in \mathcal{R}^c$ the absolute resource conflict $\Delta_k := r_k^c(S) - R_k$ (ARC), the relative resource conflict Δ_k / R_k (RRC) and the number of consuming activities $|V_k^c|$ (NCA). Concluding, the conflict resource for the decomposition of $\mathcal{S}_{\mathrm{T}}(W)$ is given by

$$k := \min\left\{k' \in \mathcal{R}^c \mid \pi_{k'} = \operatorname*{ext}_{l \in \mathcal{R}^c} \pi_l\right\},$$

where ext ∈ {min, max} determines if lower (min) or greater (max) priority values are preferred.

Thus far, the different possibilities of the branch-and-bound procedure to build the search tree have been considered. Next, we take a closer look on the procedures applied on the search nodes to improve the performance. While the applications of lower bounds and dominance rules covered in Sects. 6 and 7 are straightforward, the consistency tests described in Sect. 5 require further explanations.

Algorithm 6: General framework to apply domain-consistency tests

```
Input: Start time restriction $W$, set $\Gamma^\beta$, iteration limit $\alpha$
Output: Updated start time restriction $\gamma^\alpha_\beta(W)$

1 repeat
2     $W' := W$
3     forall $\gamma \in \Gamma^\beta$ do
4         $W := \gamma(W)$
5 until $W = W'$ or the iteration limit $\alpha$ is reached
6 return $W$
```

In general, different consistency tests are iteratively applied until any fixed point is reached. In our case, this fixed point is unique which can be deduced from Theorem 2.2 in Dorndorf et al. (2000a) due to the monotony of all consistency tests used in this work, i.e., $\gamma(W) \subseteq \gamma(W')$ holds if $W \subseteq W'$ is given. Algorithm 6 shows a procedure to apply iteratively a set Γ^β of domain-consistency tests on any start time restriction W until either the unique fixed point is detected ($W = W'$) or a maximum number of iterations α is reached. This procedure is equal to Algorithm 2.1 in Dorndorf et al. (2000a) except that an iteration limit is considered additionally. The outcome of the algorithm is denoted by $\gamma^\alpha_\beta(W)$ which is in general, due to the iteration limit, not equal to the unique fixed point. For the computational studies as shown in Sect. 10, we have examined the sets Γ^B, Γ^D and Γ^W of domain-consistency tests, where Γ^B comprises the temporal-bound and the resource-bound consistency test, Γ^D the temporal-bound and D-interval consistency test and Γ^W the temporal and the W-interval consistency test. Since we assume that the D-interval and W-interval consistency test can both be conducted either by considering all resources $k \in \mathcal{R}$ or only the resources $k \in \mathcal{R}_i$ which are demanded by activity $i \in V$, we distinguish the corresponding outcomes of Algorithm 6 by $\gamma^\alpha_\beta[\mathcal{R}]$ and $\gamma^\alpha_\beta[\mathcal{R}_i]$ for Γ^D and Γ^W, respectively. Conclusively, we have additionally examined two alternatives for the maximum number of iterations with $\alpha = 1$ and $\alpha = \infty$, where $\alpha = \infty$ implies that Algorithm 6 determines the unique fixed point with respect to Γ^β.

Algorithm 7: 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  Set $ES_i := d_{0i}$, $LS_i := -d_{i0}$ and $W_i := \{ES_i, .., LS_i\}$ for all $i \in V$
3  Apply Preprocessing on $W$
4  if $\mathcal{S}_T(W) = \emptyset$ then terminate ($\mathcal{S} = \emptyset$)
5  Set $LB^G := \min W_{n+1}$   $S := \min \mathcal{S}_T(W)$   $LB := LB^G$
6  $\Omega := \{(W, S, LB)\}$   $UB := \bar{d} + 1$
7  while $\Omega \neq \emptyset$ do
8      Remove triple $(W, S, LB)$ from stack $\Omega$
9      if $LB < UB$ then
10         Apply consistency tests on $W$
11         if $\mathcal{S}_T^{UB}(W) \neq \emptyset$ then
12             Update $S := \min \mathcal{S}_T(W)$
13             if $S \in \mathcal{S}_R$ then
14                 $S^* := S$   $UB := S^*_{n+1}$
15             else
16                 Initialize $\Lambda := \emptyset$ and select conflict resource $k \in \mathcal{R}^c$
17                 forall $i \in \{j \in V_k \mid r^u_{jk}(S_j) > 0\}$ do
18                     $W' := W$   $W'_i := W'_i \cap W_{ik}(r^u_{ik}(S_i) - 1)$
19                     if $\mathcal{S}_T^{UB}(W') \neq \emptyset$ then
20                         Set $S' := \min \mathcal{S}_T(W')$
21                         if $S' \in \mathcal{S}_R$ then
22                             $S^* := S'$   $UB := S^*_{n+1}$
23                         else
24                             Apply dominance rules on $W'$
25                             Compute lower bound $LB'$
26                             if $W'$ is not dominated and $LB' < UB$ then
27                                 Set $\Lambda := \Lambda \cup \{(W', S', LB')\}$
28                 Put all nodes from $\Lambda$ on stack $\Omega$ (ordering strategy)
29 if $UB = \bar{d} + 1$ then terminate ($\mathcal{S} = \emptyset$)
30 else return $S^*$
```

Algorithm 7 outlines the framework of the branch-and-bound procedure, where in order to improve the readability, it is assumed that a depth-first search is used and that for each explored node all child nodes are generated. This means that all other alternatives described above for the traversing and generation strategy are omitted in Algorithm 7.

In the first step of Algorithm 7, the start time restriction W is initialized where the Floyd–Warshall algorithm (Ahuja et al. 1993, Sect. 5.6) is used to determine the distance matrix D or to prove that the project network contains a cycle of positive length ($\mathcal{S} = \emptyset$) with a time complexity of $\mathcal{O}(|V|^3)$. Next, a preprocessing step is applied on the start time restriction W, where the unique fixed point of set Γ^W considering all resources is calculated, i.e., $W := \gamma_W^\infty[\mathcal{R}](W)$ is determined. In case that the preprocessing step cannot exclude the existence of a feasible schedule ($\mathcal{S}_T(W) \neq \emptyset$), the global lower bound LB^G is set to $\min W_{n+1}$, where it should be noted that due to the preprocessing step $\min W_{n+1} = \mathrm{LBD}^\pi(W)$ holds. After that, the root node given by a triple (W, S, LB) is put on stack Ω and the upper bound $\mathrm{UB} := \bar{d} + 1$ is initialized.

In each iteration, a triple (W, S, LB) is taken from stack Ω. If the corresponding node cannot be pruned due to $\mathrm{LB} \geq \mathrm{UB}$, consistency tests as described above are applied on the start time restriction W. It should be noted that for the temporal-bound and the temporal consistency test a maximum project duration of UB − 1 is assumed, whereby it is taken into account that an optimal schedule has a project duration not greater than UB. Since the resource-bound, D-interval and W-interval consistency tests are dependent on the given maximum project duration, this can increase the number of inconsistent start times, respectively. If after the application of the consistency tests there exists no W-feasible schedule with a lower project duration than UB, i.e., $\mathcal{S}_T^{UB}(W) := \widehat{\mathcal{S}}_T(W, n+1, UB-1) = \emptyset$, then the corresponding node can be pruned. Otherwise, in case that the consistency tests have removed any start time from W, schedule S is updated. If S is resource-feasible, a new best schedule has been found, schedule S is stored by $S^* := S$, and the upper bound UB for the project duration is set to S^*_{n+1}. In case that schedule S is not resource-feasible, the feasible region $\mathcal{S}_T(W)$ is decomposed as described in Sect. 3 based on the selected conflict resource $k \in \mathcal{R}^c$ corresponding to the priority rule of the branching strategy. As explained in Sect. 8, the decomposition could also be replaced by a partitioning of the feasible region. For each generated child node, which does not exclude the existence of a W-feasible schedule with a project duration lower than UB, $S' := \min \mathcal{S}_T(W')$ is determined. If S' is resource-feasible, S' is stored as the best solution and UB is updated as described above. Otherwise, dominance rules are applied on W' and the lower bound LB$'$ is calculated if W' is not dominated by another node in Ω which is no ancestor of the search node. Lower bound LB$'$ is either given by LB0$^\pi$ or LBD$^\pi(W, S'_{n+1}, UB-1)$. where in case of $LB' < UB$ the child node is stored in list Λ. After the generation of all child nodes, the nodes in list Λ are put on stack Ω corresponding to the ordering strategy, so that the child node with the best priority value is considered in the next iteration. The described procedure is iteratively conducted until the stack Ω does not contain any triple. Regarding the correctness of the enumeration scheme, Algorithm 7 returns an optimal schedule if and only if any feasible solution has been found which is given by $UB \leq \bar{d}$. Accordingly, the infeasibility of the considered instance is proven by $UB = \bar{d} + 1$ at the end of the algorithm.

10 Performance analysis

In order to evaluate the performance of our branch-and-bound procedure, we have conducted computational experiments on test instances comparing the branch-and-bound procedure with the mixed-integer linear programming (MILP) solver IBM CPLEX 12.8.0. Based on the binary linear program in Böttcher et al. (1999) for the RCPSP/π, we have developed different mathematical programs for the RCPSP/max-π which differ according to the considered type of decision variable. The different types of decision variables we have used are well known as pulse and step variables (see, e.g., Artigues 2017). Preliminary tests have shown that the program based on step variables provides the best results. Accordingly, in the following we compare our branch-and-bound procedure with the IBM CPLEX solver based on a formulation with step variables which can be stated as follows:

$$\begin{array}{llll}
\text{Minimize} & \sum_{t\in\mathcal{T}_{n+1}} t\cdot\zeta_{n+1,t} & & \\
\text{subject to} & \sum_{t\in\mathcal{T}_j} t\cdot\zeta_{jt} \geq \sum_{t\in\mathcal{T}_i} t\cdot\zeta_{it} + \delta_{ij} & ((i,j)\in E) & (2)\\
& \sum_{i\in V} r^d_{ik} \sum_{t\in Q_{ik}} (z_{it} - z_{i,t-p_i}) \leq R_k & (k\in\mathcal{R}) & (3)\\
& z_{i,t-1} \leq z_{it} & (i\in V,\ t\in\mathcal{T}_i) & \\
& z_{it} = 0 & (i\in V,\ t\in H^+ : t < ES_i) & \\
& z_{it} = 1 & (i\in V,\ t\in H^+ : t \geq LS_i) & \\
& z_{it} \in \{0,1\} & (i\in V,\ t\in H^+) &
\end{array}$$

The mathematical program for the RCPSP/max-π is a time-indexed formulation with binary decision variables z_{it} for each activity $i \in V$ and all its time-feasible start times $t \in \mathcal{T}_i := \{\mathrm{ES}_i, \ldots, \mathrm{LS}_i\}$, where z_{it} takes value 1 if and only if activity $i \in V$ starts at time t or earlier. To improve the readability, we use $\zeta_{it} := z_{it} - z_{i,t-1}$ and $H^+ := H \cup \{0, -1, \ldots, -\bar{d}\}$ in the formulation. Accordingly, the constraints of the program state that all temporal (2) and resource constraints (3) are satisfied with $Q_{ik} := \{\tau \in \mathcal{T}_i^+ \mid \tau+1 \in \Pi_k\}$ and $\mathcal{T}_i^+ := \{\mathrm{ES}_i, \ldots, \mathrm{LS}_i + p_i - 1\}$, while the remaining conditions ensure that each activity is started exactly once.

Since instances for the RCPSP/max-π are not available in the open literature, we have used self-generated instances for the performance analysis. The new instance sets are based on the well-known benchmark test sets UBO for the RCPSP/max which were generated by the instance generator ProGen/max (Schwindt 1996, 1998b) and are available via the project scheduling library PSPLIB (Kolisch and Sprecher 1997). In a first step, for each of the UBO test sets with $n = 10, 20, 50, 100, 200$ real activities, we have chosen three instances which were generated by ProGen/max with values 0.25, 0.5 and 0.75 for the so-called order strength. The order strength *OS* is an estimator for the restrictiveness of a digraph which can be seen as a [0,1]-normalized control parameter for the number of possible execution sequences of the activities of the project with $\mathrm{OS} = 0$ implicating a parallel and $\mathrm{OS} = 1$ a series digraph. Since the restrictiveness cannot efficiently be calculated, the order strength OS is used instead, where it has been shown to provide the lowest mean relative error to the restrictiveness among 40 evaluated estimators in Thesen (1977). Since the actual order strength OS' after the generation of an instance is in general not equal to the target value OS, we have chosen for each UBO test set ($n = 10, 20, 50, 100, 200$) the instance with the lowest number for which OS' deviates less than 10% from the target value $\mathrm{OS} = 0.25, 0.5, 0.75$. Concluding, the project networks of the new instance sets for the RCPSP/max-π are taken from the UBO test sets with $n = 10, 20, 50, 100, 200$ real activities which cover the processing times of the activities and all temporal constraints.

Accordingly, it remains to consider the generation of the problem parameters concerning the partially renewable resources. For the generation of the corresponding parameters, we have used the procedure described in Schirmer (1999, Sect. 10) for the instance generator ProGen/Π which is an extension of the instance generator ProGen for the RCPSP (Kolisch et al. 1995). In Schirmer (1999), three [0,1]-normalized control parameters called horizon factor (HF), cardinality factor (CF) and interval factor (IF) are used to determine Π_k for each resource $k \in \mathcal{R}$, where in line with Schirmer

(1999), each instance contains 30 partially renewable resources. The horizon factor determines an upper bound $\bar{d}^{\mathcal{R}}$ for the last period in Π_k for each resource $k \in \mathcal{R}$ with $\bar{d}^{\mathcal{R}} := \mathrm{ES}_{n+1}\,(1-\mathrm{HF}) + \bar{d} \cdot \mathrm{HF}$, where we assume that for each instance a maximum project duration $\bar{d} := \sum_{i \in V} \max(p_i, \max_{(i,j)\in E} \delta_{ij})$ is given. Depending on $\bar{d}^{\mathcal{R}}$, the cardinality factor assigns a cardinality of $|\Pi| := 2\,(1-\mathrm{CF}) + (\bar{d}^{\mathcal{R}} - 1)\,\mathrm{CF}$ to Π_k of each resource $k \in \mathcal{R}$, where an upper bound for the number of components in Π_k is directly given by $\bar{I} := \min(|\Pi|, \bar{d}^{\mathcal{R}} - |\Pi| + 1)$ (Schirmer 1999, Lemma 10.2). Finally, the interval factor determines the number of components for each resource $k \in \mathcal{R}$ by $\mathcal{I}_k := (1-\mathrm{IF}) + \bar{I} \cdot \mathrm{IF}$, so that after the assignment of $|\Pi|$ periods to $\mathcal{I}_k$ components in Π_k and the determination of the number of periods between them, the period sets Π_k of all resources $k \in \mathcal{R}$ are defined. For further details, we refer the reader to Schirmer (1999, Sect. 10). In order to control the average ratio of all resources used per real activity $i \in V^r := V \setminus \{0, n+1\}$, the resource factor (RF) is used which is defined by

$$\mathrm{RF} := \frac{1}{|V^r||\mathcal{R}|} \sum_{i \in V^r} \sum_{k \in \mathcal{R}} a_{ik} \qquad a_{ik} := \begin{cases} 1, \text{ if } r^d_{ik} > 0 \\ 0, \text{ otherwise} \end{cases},$$

where in a first step, as described in Kolisch et al. (1995), each real activity $i \in V^r$ is randomly assigned a number $|\mathcal{R}_i| \in \{\underline{a}, \ldots, \overline{a}\}$ of demanded resources ($|\mathcal{R}_i| = \sum_{k \in \mathcal{R}} a_{ik}$), which is followed by a random selection of the corresponding resource demands r^d_{ik} from set $\{\underline{r}, \ldots, \overline{r}\}$. For the generation of all instance sets for the RCPSP/max-π , we have used the parameters $\underline{a} = 5$, $\overline{a} = 25$, $\underline{r} = 1$ and $\overline{r} = 10$. Finally, the resource strength (RS) regulates the degree of scarcity of the resources by specifying the amounts of resource capacities. As all control parameters described before, the resource strength is restricted to values in [0,1] as well. Dependent on the resource strength, for each resource $k \in \mathcal{R}$ the capacity is set to $R_k := R_k^{\min}\,(1-\mathrm{RS}) + R_k^{\max} \cdot RS$ with $R_k^{\mathrm{ext}} := \sum_{i \in V_k} \mathrm{ext}\{r^c_{ik}(\tau) \,|\, \mathrm{ES}_i \le \tau \le \mathrm{LS}_i\}$ and $\mathrm{ext} \in \{\min, \max\}$ so that $\mathrm{RS} = 0$ implicates the greatest scarcity and $\mathrm{RS} = 1$ the lowest.

For each instance taken from a UBO test set with $n = 10, 20, 50, 100, 200$ real activities as described above with actual order strengths OS' with lower deviations than 10% from values $\mathrm{OS} \in \{0.25, 0.5, 0.75\}$, instances for the RCPSP/max-π have been generated based on a full factorial design with control parameters HF, CF, IF, RF, RS $\in \{0.25, 0.5, 0.75\}$ and a fixed number of 30 resources. Accordingly, we have generated for each number $n = 10, 20, 50, 100, 200$ of real activities an instance set containing 729 instances which are denoted by UBO10$^{\pi}$, UBO20$^{\pi}$, UBO50$^{\pi}$,

Table 1 Results of the performance analysis (300 s)

	$UBO10^{\pi}$		$UBO20^{\pi}$		$UBO50^{\pi}$		$UBO100^{\pi}$		$UBO200^{\pi}$	
	BnB	CPX	BnB	CPX	BnB	CPX	BnB	CPX	BnB	CPX
#nTriv	693		621		527		484		466	
#opt	534	534	500	499	145	134	79	40	79	0
#feas	534	534	578	574	486	457	465	458	466	458
#inf	159	159	40	40	3	21	0	0	0	0
#open	0	0	3	7	38	49	19	26	0	8
$\varnothing^{CPU}_{opt}$	0.040 s	0.619 s	8.076 s	27.170 s	8.022 s	55.303 s	20.681 s	68.018 s	28.271 s	–
$\varnothing^{CPU}_{inf}$	0.004 s	0.278 s	8.006 s	1.997 s	0.279 s	55.508 s	–	–	–	–

$UBO100^{\pi}$ and $UBO200^{\pi}$ in the following. To provide a benchmark test set for the RCPSP/max-π, we made these test sets available online.[1]

The computational experiments have been conducted on a PC with an Intel Core i7-8700 3.2 GHz CPU and 64 GB RAM under Windows 10 with a time limit of 300 s. The branch-and-bound algorithm and the binary linear program for the RCPSP/max-π were both coded in C++ and compiled with the 64-bit Visual Studio 2017 C++ compiler where we used the IBM OPL C++ interface for the linear program. To solve the program, we have applied the MILP solver IBM CPLEX 12.8.0 restricted to a single thread in order to ensure a fair comparison with the branch-and-bound algorithm which is conducted on a single thread as well.

Table 1 shows the results of the computational performance analysis with a time limit of 300 s based on the settings given in Table 3 which are discussed later on. For each instance set, the results of the branch-and-bound algorithm (BnB) and the MILP solver IBM CPLEX 12.8.0 (CPX) are compared. In the first row, the number of non-trivial instances (#nTriv) of the corresponding instance set is given where in line with Alvarez-Valdes et al. (2008) an instance is called non-trivial if and only if schedule ES is not resource-feasible. Since for each trivial instance an optimal solution can efficiently be determined, all results in Table 1 are restricted to non-trivial instances only. The following rows show the number of instances which were solved to optimality (#opt), for which a feasible solution was found (#feas), the infeasibility could be proved (#inf) or the instance status (feasible or infeasible) remained open (#open). Conclusively, the last two rows list the average used CPU time in seconds over all instances which were solved to optimality ($\varnothing^{CPU}_{opt}$) and which were proved to be infeasible ($\varnothing^{CPU}_{inf}$).

The results in Table 1 indicate that BnB outperforms CPX in finding for more instances feasible solutions and in proving more solutions to be optimal over all instance sets with more than ten activities, where the differences even increase with the instance size. As a consequence, BnB is able to determine the status for more instances as well. Additionally, it can be seen that BnB has also an advantage over CPX regarding the average used CPU time over all instances which were solved to optimality ($\varnothing^{CPU}_{opt}$). In contrast, CPX dominates BnB in the sense of proving either more

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

Table 2 Comparison of the feasible solutions between BnB and CPX (300 s)

Instance set	$\#^{\cup}_{\text{feas}}$	$\#^{\cap}_{\text{feas}}$	$\#^{<}_{\text{feas}}$	$\#^{>}_{\text{feas}}$	$\#^{\cap,\text{nv}}_{\text{feas}}$	$\#^{<}$	$\#^{=}$	$\#^{>}$	$\varnothing^{\Delta,\text{abs}}_{\text{CPX}}$	$\varnothing^{\Delta,\text{rel}}_{\text{CPX}}$
UBO20$^{\pi}$	579	573	5	1	95	25	42	28	– 0.92	– 0.29%
UBO50$^{\pi}$	488	455	31	2	332	310	7	15	– 86.60	– 17.09%
UBO100$^{\pi}$	466	457	8	1	417	417	0	0	– 412.71	– 35.09%
UBO200$^{\pi}$	466	458	8	0	458	458	0	0	– 1056.86	– 42.98%

instances to be infeasible or using less average CPU time for the proof considering instance sets UBO20$^{\pi}$ and UBO50$^{\pi}$.

The results of the performance analysis in Table 1 are supplemented by Table 2 which provides a closer look at the feasible solutions of BnB and CPX. The first four columns of Table 2 investigate to what extent the solution procedures are able to find feasible solutions for different instances. For this, the columns list the number of feasibly solved instances by at least one solution procedure ($\#^{\cup}_{\text{feas}}$) and by both procedures ($\#^{\cap}_{\text{feas}}$), followed by the number of instances for which only BnB ($\#^{<}_{\text{feas}}$) or CPX ($\#^{>}_{\text{feas}}$) could find a solution. From the first part of Table 2, it can be seen that the proportion of feasible instances which could only be shown to be feasible by BnB is much greater than the proportion of CPX over all instance sets. The second part of Table 2 addresses the quality of the solutions for all instances which could be solved feasibly, but not verified as optimal by both solution procedures ($\#^{\cap,\text{nv}}_{\text{feas}}$). These instances are segmented in the following columns into instances for which BnB could find a better solution than CPX ($\#^{<}$), both procedures provided a solution with an equal project duration ($\#^{=}$) or CPX was able to detect a better solution than BnB ($\#^{>}$). Finally, the last two columns list the average deviations between the objective function values of BnB and CPX. Given the corresponding objective function values by S^{BnB}_{n+1} and S^{CPX}_{n+1}, the first column shows the average absolute deviation $\Delta^{\text{abs}}_{\text{CPX}} := S^{\text{BnB}}_{n+1} - S^{\text{CPX}}_{n+1}$ over all considered instances ($\varnothing^{\Delta,\text{abs}}_{\text{CPX}}$) and the second column the average relative deviation $\Delta^{\text{rel}}_{\text{CPX}} := \Delta^{\text{abs}}_{\text{CPX}}/S^{\text{CPX}}_{n+1}$ to the objective function value of CPX ($\varnothing^{\Delta,\text{rel}}_{\text{CPX}}$). Table 2 shows that on average the quality of the solutions of BnB is better than the solutions of CPX over all instance sets regarding both the absolute and relative deviation. It can also be observed that the average deviations strongly increase with the instance size and that BnB determines for much more instances a feasible solution with a better objective function value than CPX.

The settings of BnB we have used for the computational experiments, dependent on the instance size, are given in Table 3. The listed strategies and components applied on an instance set can be seen as the setting with the best balance between the number of instances which are solved to optimality and whose status remains open among all settings we have tested. The terms used in Table 3 are in line with the descriptions in Sect. 9, except for some additional specifications we discuss in the following. In the first row, the values in brackets represent the predefined time span for the scattered-path search until any of all not completely explored nodes with lowest level in the search tree is considered next. Furthermore, the values in brackets state the maximum number of child nodes allowed to be generated in each exploration step for the generation strategy

Table 3 Settings of the branch-and-bound algorithm for the performance analysis

	UBO10$^{\pi}$	UBO20$^{\pi}$	UBO50$^{\pi}$	UBO100$^{\pi}$	UBO200$^{\pi}$
Traversing strategy	DFS	SPS^{+} [2 s]	SPS^{+} [5 s]	SPS^{+} [15 s]	SPS^{+} [15 s]
Branching strategy	NCA (min)	NCA (min)	DST (max)	DST$^{\mathrm{I}}$ (max)	DST$^{\mathrm{I}}$ (max)
Generation strategy	all	all	all	restrCL [10]	restrCL [10]
Ordering strategy	LB (min)	LB (min)	LB-DST (min)	LB-DST$^{\mathrm{I}}$ (min)	LB-DST$^{\mathrm{I}}$ (min)
Consistency tests	$\gamma_B^{\infty}, \gamma_W^{1}[\mathcal{R}_i, 10]$	$\gamma_B^{\infty}, \gamma_W^{1}[\mathcal{R}_i, 10]$	$\gamma_B^{\infty}, \gamma_D^{1}[\mathcal{R}_i]$	$\gamma_B^{\infty}, \gamma_D^{1}[\mathcal{R}_i, 5]$	γ_B^{∞}
Lower bound	LBD$^{\pi}$	LBD$^{\pi}$	LBD$^{\pi}$	LB0$^{\pi}$	LB0$^{\pi}$
Partitioning	x	x	–	–	–

and the greatest search tree level on which the corresponding set of consistency tests is applied (Consistency tests). For the branching and ordering strategy, the symbol ext $\in$ {min, max} which determines the ordering of the priority values is given in parentheses. Finally, in the last row, "x" indicates that the enumeration scheme is conducted based on the concepts described in Sect. 8, whereas "–" stands for the application of the enumeration described in Sect. 3.

From Table 3, it can be seen that for small instances (UBO10$^{\pi}$ and UBO20$^{\pi}$) the partitioning of the feasible region of each search node is beneficial for the performance, whereas computational experiments on greater instances have shown that this procedure leads to a rather bad performance regarding the number of instances for which the status can be determined. The reason for this could be caused by the fact that due to the partitioning, each part of the feasible region $\mathcal{S}$ can just be reached by at most one path in the enumeration tree which most likely results in a decrease in the probability to find a feasible solution at all. Furthermore, Table 3 shows that it is important for the performance to decrease the intensity of consistency tests, to restrict the number of generated child nodes in each exploration step and to invest less computational effort on the calculation of lower bounds with the increase in the instance size. It should also be noted that the scattered-path search is already preferable to choose for instance sets with more than ten activities and that the most promising priority values and their orderings are dependent on the instance size as well.

Computational tests have shown that the application of dominance rules can improve the performance of the branch-and-bound procedure for small instances as well if the enumeration procedure is conducted without partitioning. To show this, Table 4 compares the different techniques covered in this paper to avoid redundancies in the search tree for instance set UBO10$^{\pi}$ with a time limit of 300 s. In the first row, the results of BnB corresponding to the settings given in Table 3 without partitioning are given, where the last columns show the average number of completely explored nodes per instance ($\varnothing^{\mathrm{expl}}_{\mathrm{nodes}}$) and the total used CPU time over all instances (t_{cpu}). The following rows list the results if either only the $\bar{U}$-dominance rule, the W-dominance rule or both dominance rules ($\bar{U}/W$) are applied in addition. From Table 4, it can be seen that both dominance rules are able to reduce the average number of explored nodes per instance and the total used CPU time accompanied by an increase in optimally solved and infeasible proved instances where the W-dominance rule shows a

Table 4 Comparison of techniques to avoid redundancies for instance set UBO10$^{\pi}$ (300 s)

	#opt	#feas	#inf	#open	$\varnothing^{\text{expl}}_{\text{nodes}}$	t_{cpu}
w/o Partitioning	507	534	157	2	219,439	9745 s
$\bar{U}$-dominance	521	534	158	1	100,451	5242 s
W-dominance	531	534	159	0	27,879	1824 s
$\bar{U}/W$-dominance	531	534	159	0	24,060	1701 s
Partitioning	534	534	159	0	81	23 s

Table 5 Impact of components on the performance for instance set UBO10$^{\pi}$ (300 s)

	#opt	#feas	#inf	#open	$\varnothing^{\text{CPU}}_{\text{opt}}$	$\varnothing^{\text{CPU}}_{\text{inf}}$	t_{cpu}
BnB (basic version)	345	526	107	60	9.310 s	7.526 s	76,317 s
+Preprocessing	415	530	157	6	5.485 s	0.141 s	38,598 s
+LBD$^{\pi}$	438	532	157	4	6.387 s	0.022 s	32,201 s
+Consistency tests	507	534	157	2	2.061 s	0.004 s	9745 s
+$\bar{U}$-dominance	521	534	158	1	1.949 s	0.167 s	5242 s
+W-dominance	531	534	159	0	1.483 s	0.084 s	1701 s
+Partitioning	534	534	159	0	0.040 s	0.004 s	23 s

better performance. Furthermore, it can be observed that the application of both dominance rules shows a slightly better performance where three instances still remain without an optimality proof. The last row presents the results if the feasible region of each node in the search tree is partitioned as described in Sect. 8. These results demonstrate the dominance of the partitioning technique over the dominance rules with a tremendous decrease in the average number of explored nodes per instance and the total used CPU time. Similar results could be observed for instance set UBO20$^{\pi}$, whereas for greater instances neither the partitioning technique nor the dominance rules were able to improve the performance.

Next, the impact of the different components of the branch-and-bound procedure on the performance should be illustrated. For this, Table 5 shows the results of the branch-and-bound procedure based on the search strategy and different combinations of the components given in Table 3 for instance set UBO10$^{\pi}$ with a time limit of 300 s. The first row provides the results of the basic version of the branch-and-bound procedure, which means that the enumeration is done without partitioning and only the lower bound LB0$^{\pi}$ is used. The following rows show the results in case that the given component is applied in addition where it can be observed that each added component improves the performance. Conclusively, it should be mentioned that similar results are obtained for greater instances as well.

Finally, we compare our branch-and-bound algorithm with the only available exact solution procedure for partially renewable resources which is given in Böttcher et al. (1999) for the RCPSP/π (BOT). For this, Table 6 shows the results of a performance analysis conducted on test sets with 10, 20, 30 and 40 real activities (j10, j20, j30, j40) and 30 partially renewable resources, respectively, which have been generated by

Table 6 Comparison with the BnB algorithm in Böttcher et al. (1999)

	j10		j20		j30		j40	
	BnB	BOT	BnB	BOT	BnB	BOT	BnB	BOT
#nTriv	808		565		453		386	
#opt	807	758	551	339	414	250	333	169
#feas	807	797	565	542	453	431	385	352
#inf	1	0	0	0	0	0	0	0
#open	0	11	0	23	0	22	1	34
$\varnothing^{\text{CPU}}_{\text{opt}}$	10 ms	n.a.	87 ms	n.a.	206 ms	n.a.	329 ms	n.a.
$\varnothing^{\text{CPU}}_{\text{inf}}$	1 ms	–	–	–	–	–	–	–

ProGen/Π. The results for BOT in Table 6 are taken from Schirmer (1999, Sect. 10.4), where BOT was implemented in C and tested on an IBM RS/6000 workstation with 66 MHz under AIX. For the comparison, we scaled the time limit by a factor of 50 corresponding to the clock pulse ratio of the different workstations ($3,200/66 \approx 48.5$) so that we used time limits of 6 (6, 12, 24) s for BnB while 300 (300, 600, 1200) s were chosen for BOT for instance set j10 (j20, j30, j40). It should be noted that nine instances of test set j10 which were proved to be infeasible by BOT could not be provided to us, so that they are not part of the comparison. Conclusively, Table 6 shows the great dominance of BnB which has been applied with the settings for instance set UBO20$^{\pi}$ from Table 3.

11 Conclusions

We have considered the resource-constrained project scheduling problem with partially renewable resources and general temporal constraints with the objective to minimize the project duration, which to the best of our knowledge has not been considered in the open literature so far. For this problem, we have presented a branch-and-bound procedure whose enumeration scheme is based on a stepwise reduction in permitted resource usages by the activities of the project. To enhance the performance of the solution procedure, we have developed consistency tests, lower bounds and dominance rules whose efficiency could be confirmed by computational experiments. Furthermore, it could be shown that the avoidance of redundancies in the search tree, obtained by an adaptation of the enumeration scheme, significantly improves the performance for small instances. A comparison with the mixed-integer linear programming solver IBM CPLEX 12.8.0 on adaptations of benchmark test sets from literature could reveal the great dominance of the branch-and-bound procedure if feasible instances are considered. In contrast, it turned out that the solver IBM CPLEX 12.8.0 is better suited to prove instances to be infeasible. Finally, the good performance of the branch-and-bound procedure could also be confirmed by a comparison with the only exact solution procedure for the RCPSP/π.

As the results of the computational study indicate, there is a great need for efficient heuristics for the RCPSP/max-π. In this context, it could be an interesting field for future research to develop heuristics which are based on the temporal planning procedures and consistency tests presented in this work. Furthermore, the investigation of alternative lower bounds and consistency tests seems to be a topic of great interest as well.

Acknowledgement Open Access funding provided by Projekt DEAL.

References

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

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

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

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

Artigues C (2017) On the strength of time-indexed formulations for the resource-constrained project scheduling problem. Oper Res Lett 45(2):154–159

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

Bartusch M, Möhring RH, Radermacher FJ (1988) Scheduling project networks with resource constraints and time windows. Ann Oper Res 16(1):201–240

Bianco L, Caramia M (2012) An exact algorithm to minimize the makespan in project scheduling with scarce resources and generalized precedence relations. Eur J Oper Res 219(1):73–85

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

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

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

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

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

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

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

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

Franck B, Neumann K, Schwindt C (2001a) Project scheduling with calendars. OR Spektrum 23(3):325–334

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

Klein R, Scholl A (1999) Computing lower bounds by destructive improvement: an application to resource-constrained project scheduling. Eur J Oper Res 112(2):322–346

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

Kolisch R, Sprecher A (1997) PSPLIB: a project scheduling problem library. Eur J Oper Res 96(1):205–216

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

Kreter S (2016) Projektplanung mit Kalendern (project scheduling with calendars): Struktureigenschaften und Lösungsmethoden (structural characteristics and solution procedures). Shaker, Aachen

Kreter S, Rieck J, Zimmermann J (2016) Models and solution procedures for the resource-constrained project scheduling problem with general temporal constraints and calendars. Eur J Oper Res 251(2):387–403

Murty KG (1968) An algorithm for ranking all the assignments in order of increasing cost. Oper Res 16(3):682–687

Neumann K, Schwindt C (1995) Projects with minimal and maximal time lags: construction of activity-on-node networks and applications. Report WIOR-447, University of Karlsruhe

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

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

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

Schutt A, Feydy T, Stuckey PJ, Wallace MG (2013) Solving RCPSP/max by lazy clause generation. J Sched 16(3):273–289

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

Schwindt C (1998a) A branch-and-bound algorithm for the resource-constrained project duration problem subject to temporal constraints. Report WIOR-544, University of Karlsruhe

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

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

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

Thesen A (1977) Measures of the restrictiveness of project networks. Networks 7(3):193–208