Technical Report AC-TR-17-001

first version: March 2017
updated: September 2017

An Iterative Time-Bucket Refinement Algorithm for a High Resolution Resource-Constrained Project Scheduling Problem

Martin Riedler, Thomas Jatschka, Johannes Maschler, and Günther R. Raidl

An Iterative Time-Bucket Refinement Algorithm for a High Resolution Resource-Constrained Project Scheduling Problem

Martin Riedler, Thomas Jatschka,
Johannes Maschler, and Günther R. Raidl
Institute of Computer Graphics and Algorithms, TU Wien, Austria
{maschler|riedler|raidl}@ac.tuwien.ac.at, jatschka.thomas@gmail.com

We consider a resource-constrained project scheduling problem originating in particle therapy for cancer treatment, in which the scheduling has to be done in high resolution. Traditional mixed integer linear programming techniques such as time-indexed formulations or discrete-event formulations are known to have severe limitations in such cases, i.e., growing too fast or having weak linear programming relaxations. We suggest a relaxation based on partitioning time into so-called time-buckets. This relaxation is iteratively solved and serves as basis for deriving feasible solutions using heuristics. Based on these primal and dual solutions and bounds the time-buckets are successively refined. Combining these parts we obtain an algorithm that provides good approximate solutions soon and eventually converges to an optimal solution. Diverse strategies for performing the time-bucket refinement are investigated. The approach shows excellent performance in comparison to the traditional formulations and a metaheuristic.

Keywords. Resource-Constrained Project Scheduling, Time-Bucket Relaxation, Mixed Integer Linear Programming, Matheuristics, Particle Therapy

1 Introduction

Scheduling problems arise in a variety of practical applications. Prominent examples are job shop or project scheduling problems that require a set of activities to be scheduled over time. The execution of the activities typically depends on certain resources of limited availability and diverse other restrictions such as precedence constraints. The goal is to find a feasible schedule that minimizes some objective function like the makespan. In certain cases, scheduling has to be done in a very fine-grained way, i.e., in high resolution, using, e.g., seconds or even milliseconds as unit of time.

Classical mixed integer linear programming (MILP) formulations are known to struggle under these conditions. On the one hand, time discretized models provide strong linear programming (LP) bounds but grow too quickly with the instance size due to the fine time discretization. Event-based and sequencing-based models on the other hand typically have trouble as a result of their weak LP bounds.

In the following, we focus on problems with a large, very fine-grained scheduling horizon and consider a simplified scheduling problem arising in the context of modern particle therapy used for cancer treatment. The problem is motivated by a real world patient scheduling scenario at the recently founded cancer treatment center MedAustron located in Wiener Neustadt, Austria[1]. The tasks involved in providing a given set of patients with their individual particle treatments shall be scheduled in such a way that given precedence constraints with minimum and maximum time lags are respected. Each task needs certain resources for its execution. One of the resources is the particle beam, which is particularly scarce as it is required by every treatment and shared between several treatment rooms. The motivation therefore is to exploit in particular the availability of the beam as good as possible by suitably scheduling all activities in high time resolution. Ideally, the beam is switched immediately after an irradiation has taken place in one room to another room where the next irradiation session starts without delay.

Our goal is to minimize the makespan. This objective emerges from the practical scenario as tasks need to be executed as densely as possible to avoid idle time within the day as well as to allow treating as many patients as possible within the operating hours. However, makespan minimization is clearly an abstraction from the real world scenario where more specific considerations need to be taken into account. In the terminology of the scientific literature in scheduling, the considered problem corresponds to a resource-constrained project scheduling problem with minimum and maximum time lags.

In this work, we introduce the simplified intraday particle therapy patient scheduling problem (SI-PTPSP) and present for it a discrete-event formulation and a time-indexed formulation as reference models. We propose a time-bucket relaxation (TBR) and prove some theoretical properties. In the main part, we deal with the iterative time-bucket refinement algorithm (ITBRA) that aims at closing the gap between dual solutions obtained by solving TBR based on iteratively refined bucket partitionings and heuristically determined primal solutions exploiting dual solutions. Various strategies for refining the bucket partitioning are suggested. Experimental results clearly indicate the superiority of the new matheuristic approach over the reference MILP models as well as a basic greedy randomized adaptive search procedure (GRASP).

The remainder of the article is organized as follows. In Section 2 we provide a formal definition of the SI-PTPSP. Then, we review the related literature. In the following section we provide two reference MILP formulations. The main part consists of the description of TBR and its properties in Section 5 and the presentation of ITBRA in

[1]https://www.medaustron.at

Section 6. We provide the fundamental iterative framework with its specifically used sub-algorithms, which are the gap closing heuristic, the activity block construction heuristic, a GRASP metaheuristic, and the investigated bucket refinement strategies. Further implementation details such as preprocessing procedures are covered in Section 7. Finally, we discuss computational experiments conducted on two sets of benchmark instances in Section 8, before concluding and giving an outlook on promising future research directions in Section 9.

2 Simplified Intraday Particle Therapy Patient Scheduling Problem

The simplified intraday particle therapy patient scheduling problem (SI-PTPSP) is defined on a set of activities $A = \{1, \dots, \alpha\}$ and a set of unit-capacity resources $R = \{1, \dots, \rho\}$. Each activity $a \in A$ is associated with a processing time $p_a \in \mathbb{N}_{>0}$, a release time $t^{\mathrm{r}}_a \in \mathbb{N}_{\geq 0}$, and a deadline $t^{\mathrm{d}}_a \in \mathbb{N}_{\geq 0}$ with $t^{\mathrm{r}}_a \leq t^{\mathrm{d}}_a$. For its execution an activity $a \in A$ requires a subset $Q_a \subseteq R$ of the resources. Activities need to be executed without preemption. The considered set of time slots $T = \{T^{\min}, \dots, T^{\max}\}$ is derived from the properties of the activities as follows: $T^{\min} = \min_{a \in A} t^{\mathrm{r}}_a$ and $T^{\max} = \max_{a \in A} t^{\mathrm{d}}_a - 1$. We denote by $Y_a(t)$ the set of time points during which activity $a \in A$ executes when starting at time t, i.e., $Y_a(t) = \{t, \dots, t + p_a - 1\}$. To model dependencies among the activities, we consider a directed acyclic precedence graph $G = (A, P)$ with $P \subset A \times A$. Each arc $(a, a') \in P$ is associated with a minimum and a maximum time lag $L^{\min}_{a,a'}, L^{\max}_{a,a'} \in \mathbb{N}_{\geq 0}$ with $L^{\min}_{a,a'} \leq L^{\max}_{a,a'}$. For each resource $r \in R$ a set of availability time windows $W_r = \bigcup_{w=1,\dots,\omega_r} W_{r,w}$ with $W_{r,w} = \{W^{\mathrm{start}}_{r,w}, \dots, W^{\mathrm{end}}_{r,w}\} \subseteq T$ is given. Resource availability windows are non-overlapping and ordered according to starting time $W^{\mathrm{start}}_{r,w}$. In accordance with the resource availabilities and the precedence relations among the activities, we can deduce for each activity a set of feasible starting times, denoted by $T_a \subseteq \{t^{\mathrm{r}}_a, \dots, t^{\mathrm{d}}_a - p_a\}$; for details on the computation of this set see Section 7.1.

A feasible solution S (also called schedule) to SI-PTPSP is a vector of values $S_a \in T_a$ assigning each activity $a \in A$ a starting time within its release time and deadline s.t. the availabilities of the required resources and all precedence relations are respected. The goal is to find a feasible solution having minimum makespan.

Using the notation introduced in Brucker et al. [1999] our problem can be classified as $\mathrm{PS}m, \cdot, 1|r_j, d_j, temp|C_{\max}$.

Computational Complexity Lawler and Lenstra [1982] have shown that finding a solution for the non preemptive single machine scheduling problem with deadlines and release times ($1|r_j|C_{\max}$ according to the notation by Graham et al. [1979]) is $\mathcal{NP}$-hard. We can easily reduce an instance of $1|r_j|C_{\max}$ to an instance of SI-PTPSP by assigning the same resource to each activity of the $1|r_j|C_{\max}$ instance. Processing times, release times, and deadlines of the activities remain unchanged. Since there are no precedence constraints in $1|r_j|C_{\max}$, the set of precedence arcs is empty. Consequently, SI-PTPSP is $\mathcal{NP}$-hard.

3 Related Work

In this section, we discuss the related work relevant for our contribution. We start with a brief overview of resource-constrained project scheduling problems (RCPSPs). Afterwards, we review the derivation of dual bounds for such scheduling problems. Then, we give a short introduction on matheuristics applied in this domain. Finally, we review previous work that is important from the methodological point of view, i.e., that deals with time-buckets or similar aggregation techniques.

3.1 Resource-Constrained Project Scheduling

The resource-constrained project scheduling problem (RCPSP) considers scheduling of a project subject to resource and precedence constraints, where a project is represented by a graph with each node being an activity of the project. Precedence relations between activities are represented as directed edges between the nodes. The RCPSP is a well studied problem with many extensions and variations. SI-PTPSP is a combination of multiple such extensions: We use minimum and maximum time lags, release times and deadlines, and dedicated renewable resources. For a detailed description of those terms and a broader overview of RCPSP variants we refer to Hartmann and Briskorn [2010].

There exists a wide range of exact and heuristic approaches for solving the RCPSP and its extensions, for an overview see Brucker et al. [1999], Neumann et al. [2003], and Artigues et al. [2008]. Here we specifically want to focus on exact approaches. Often used are branch-and-bound (B&B) algorithms (Demeulemeester and Herroelen [1997], Bianco and Caramia [2012]) and MILP techniques. However, also constraint programming (CP), SAT, and combinations thereof gained importance, e.g., Berthold et al. [2010]. For our work we are primarily interested in MILP-based approaches and thus focus on them in the following.

A well-known technique are so-called time-indexed models, see Artigues [2017]. The classical variant uses binary variables for each time slot representing the start of an activity. In addition, there are also so-called step-based formulations, in which variables indicate if an activity has started at or before a certain time instant. This might lead to a more balanced B&B tree. Both variants typically provide strong LP bounds but struggle with larger time horizons due to the related model growth.

Also quite well-known are event-based formulations. Koné et al. [2011] and Artigues et al. [2013] provide an extensive overview. These models are based on a set of ordered events to which activity starts and ends need to be assigned, allowing modeling starting times as continuous variables. On/Off event-based formulations use the same idea but require even fewer variables. These models are usually independent of any time discretization and the time horizon but feature significantly weaker LP bounds compared to time-indexed models.

There also exist formulations combining continuous-time and discrete-time formulations, so-called mixed-time models, see Westerlund et al. [2007], Baydoun et al. [2016]. Further MILP techniques make use of exponentially sized models and apply advanced methods such as column generation, Lagrangian decomposition, or Benders decomposition, see, e.g., Hooker [2007].

3.2 Dual Bounds for Scheduling Problems

The most common approach for deriving lower bounds is based on solving LP relaxations, often strengthened by cutting plane methods. This technique is widely applicable but often provides only weak bounds.

Also rather well-known are algorithms based on Lagrangian relaxation, see Fisher [1973]. The basic idea is to relax a set of complex constraints by adding corresponding penalty terms to the objective function to simplify the model. Its strong reliance on a suitable problem structure limits the applicability of this technique. For an application to the RCPSP see Bianco and Caramia [2011b].

Other techniques to obtain dual bounds are less common. Li et al. [2015] consider a dual heuristic for MILP. For some nodes of the B&B tree, the heuristic attempts to improve the current dual bound by computing relaxations based on simply dropping, dualizing, or aggregating constraints. The heuristic uses dual variables and slack variables of the LP solution in order to decide which constraints to relax.

Apart from such general approaches, there are some works that consider problem specific methods. In the RCPSP context this includes, among others, Bianco and Caramia [2011a], Carlier et al. [to appear], and Dupin and Talbi [2016].

3.3 Matheuristics for Scheduling Problems

Matheuristics are a combination of mathematical programming techniques and metaheuristics, see Maniezzo et al. [2010]. The idea of matheuristics is to either improve the metaheuristic by exploiting mathematical programming techniques or to improve the mathematical programming technique with the robustness and time efficiency of the metaheuristic.

So far, Matheuristics have only been rarely considered for tackling the RCPSP. Palpant et al. [2004] present an approach based on large neighborhood search. Subproblems are generated dynamically and solved using MILP, CP, or a heuristic approach.

Further matheuristic approaches can be found in terms of the multi-mode resource-constrained multi-project scheduling problem (MRCMPSP). This is an extension of the RCPSP in which each activity is associated with a set of modes that decide the processing time and resource demand. Artigues and Hebrard [2013] solve the MRCMPSP with an algorithm consisting of four phases. In the first phase initial modes are assigned to each activity using MILP. Phases two and three generate a schedule based on the modes assigned to the activities using CP. The last phase uses a large neighborhood search procedure to improve the schedule by changing the modes of some activities. CP is used to solve the subproblems. Phases two to four are repeated until a termination criterion is met. Toffolo et al. [2016] solve the problem using a decomposition-based matheuristic. After fixing execution modes, the problem is decomposed into time windows that are solved using MILP models. Finally, a hybrid local search is employed to improve the obtained solutions.

Moreover, there are resemblances to Benders and Lagrangian-based techniques, e.g., Maniezzo and Mingozzi [1999], Möhring et al. [2003].

3.4 Time Aggregation Models

Note that the contributions mentioned in this section stay in contrast to a more common approach in which the time-discretization is coarsened in order to possibly obtain feasible but also less precise solutions, which are in general not optimal for the original problem. The approaches discussed here are characterized by iteratively refining a relaxation of the original problem until a provably optimal solution is found.

Boland et al. [2017] consider such an approach for the countinuous time service network design problem (CTSNDP). The authors solve the problem using a time-expanded network. Initially, only a partially time-expanded network is considered to avoid the substantial size of the complete network. The MILP model associated with the reduced network constitutes a relaxation to the original problem. If the optimal solution to this relaxation turns out to be feasible w.r.t. the original problem, the algorithm terminates. Otherwise, the partially time-expanded network is extended based on the current solution to obtain a more refined model. Iteratively applying this approach converges to an optimal solution due to the finite size of the full time-expanded network.

A different type of relaxation is to partition the given time horizon into subsets. Such approaches are presented by Bigras et al. [2008], Baptiste and Sadykov [2009], and Boland et al. [2016] for single machine scheduling problems. Iterative approaches based on these techniques have been primarily considered in terms of routing problems. Wang and Regan [2002] and Wang and Regan [2009] consider such an algorithm for the traveling salesman problem with time windows (TSPTW). First, the time windows of each node are partitioned into subsets. Then, for a given time window partitioning a lower bound and an upper bound are calculated, using an underconstrained MILP model and an overconstrained one. If the gap between lower and upper bound is not sufficiently small, the scheduling horizon gets further refined and the problem is solved anew.

Another algorithm of this type has been considered by Macedo et al. [2011] for solving the vehicle routing problem with time windows and multiple routes (MVRPTW). They solve a relaxation which is modeled as a network flow s.t. nodes of the graph correspond to time instants. The idea of the initial relaxation is to aggregate several time instants into each node. If the solution to the relaxation turns out to be infeasible w.r.t. the original problem, the current time discretization is locally refined by considering further time instants individually, i.e., by disaggregating nodes.

Dash et al. [2012] combine the ideas of Wang and Regan [2002] and Bigras et al. [2008] in order to solve the TSPTW. The time windows of the nodes are partitioned into buckets using an iterative refinement heuristic. Refinement decisions are based on the solution to the current LP relaxation. Afterwards, the resulting formulation is turned into an exact approach by adding valid inequalities and solved using branch-and-cut (B&C). In each node of the B&B tree a primal heuristic is applied using the reduced costs of the variables of the current LP solution.

Recently, Clautiaux et al. [2017] introduced an approach that is more generally applicable to problems that can be modeled as minimum-cost circulation problems with linking bound constraints. The proposed algorithm projects the original problem onto an aggregated approximate one. This aggregated model is iteratively refined until a provably optimal solution is found. Experiments have been conducted on a routing problem and a cutting-stock problem.

4 Reference MILP Models

In this section, we present two MILP models for SI-PTPSP following classical approaches: a discrete-event formulation (DEF) and a time-indexed formulation (TIF). Both serve as reference formulations to which we will compare our iterative time-bucket refinement algorithm (ITBRA).

4.1 Discrete Event Formulation

The discrete-event formulation (DEF) is based on the idea of considering certain events that need to be ordered and for which respective times need to be found, see also model SEE in Artigues et al. [2013]. Resource constraints then only have to be checked at the times associated with these events.

In regard to our problem, the considered events are the start and the end of each activity (activity events), and times at which the availability of a resource changes (resource events). To simplify the model, we transform all resource events into activity events by introducing a new artificial activity for each period during which a resource $r \in R$ is unavailable. To this end, we create a new activity for each maximal interval in $T \setminus W_r$ requiring the resource where the processing time is the length of the interval, and the release time and the deadline are the start and the end of the interval, respectively. Then, we define a new set of activities A' being the union of A and the artificial activities; let $\alpha' = |A'|$. Consequently, we denote by $K = \{1, \ldots, 2\alpha'\}$ the set of chronologically ordered events.

To state the model we use binary variables $x_{a,k}$ that are one if event $k \in K$ is the start of activity $a \in A$ and zero otherwise. Similarly, binary variables $y_{a,k}$ indicate whether event k is the end of activity a. Variables E_k represent the time assigned to each event k. The starting times of the activities $a \in A'$ are modeled using variables S_a. Finally, binary variables $D_{r,k}$ are one if resource $r \in R$ is used by any activity immediately after event k and zero otherwise, and variable MS denotes the makespan.

$$\min MS \tag{1}$$

$$S_a + p_a \leq MS \qquad \forall a \in A \tag{2}$$

$$S_{a'} - S_a \geq p_a + L^{\min}_{a,a'} \qquad \forall (a, a') \in P \tag{3}$$

$$S_{a'} - S_a \leq p_a + L^{\max}_{a,a'} \qquad \forall (a, a') \in P \tag{4}$$

$$\sum_{k \in K} x_{a,k} = 1 \qquad \forall a \in A' \tag{5}$$

$$\sum_{k \in K} y_{a,k} = 1 \qquad \forall a \in A' \tag{6}$$

$$\sum_{a \in A'} (x_{a,k} + y_{a,k}) = 1 \qquad \forall k \in K \tag{7}$$

$$E_{k-1} \leq E_k \qquad \forall k \in K \setminus \{1\} \tag{8}$$

$$E_k - M^{(9)}_{a,k}(1 - x_{a,k}) \leq S_a \qquad \forall k \in K, a \in A' \tag{9}$$

$$E_k + M^{(10)}_{a,k}(1 - x_{a,k}) \geq S_a \qquad \forall k \in K, a \in A' \tag{10}$$

$$E_k - M^{(11)}_{a,k}(1 - y_{a,k}) \leq S_a + p_a \qquad \forall k \in K, a \in A' \tag{11}$$

$$E_k + M^{(12)}_{a,k}(1 - y_{a,k}) \geq S_a + p_a \qquad \forall k \in K, a \in A' \tag{12}$$

$$D_{r,0} = \sum_{a \in A' : r \in Q_a} x_{a,0} \qquad \forall r \in R \tag{13}$$

$$D_{r,k} = D_{r,k-1} + \sum_{a \in A' : r \in Q_a} x_{a,k} - \sum_{a \in A' : r \in Q_a} y_{a,k} \qquad \forall k \in K \setminus \{1\}, r \in R \tag{14}$$

$$D_{r,k} \leq 1 \qquad \forall k \in K, r \in R \tag{15}$$

$$t^{\mathrm{r}}_a \leq S_a \leq t^{\mathrm{d}}_a - p_a \qquad a \in A' \tag{16}$$

$$MS, E_k, D_{r,k} \geq 0 \qquad \forall k \in K, a \in A', r \in R \tag{17}$$

$$x_{a,k}, y_{a,k} \in \{0, 1\} \qquad \forall k \in K, a \in A' \tag{18}$$

Inequalities (2) are used for determining the makespan. Precedence relations are enforced by Inequalities (3) and (4). According to Equalities (5) and (6) each activity starts and ends at precisely one event. Equalities (7) ensure that each event is assigned to either exactly one starting time or exactly one ending time of an activity. Events are ordered chronologically by Inequalities (8). Starting times of activities are linked to the corresponding start events by Inequalities (9) and (10). Similarly, Inequalities (11) and (12) link the event at which an activity a ends to the time at which the activity ends. Big-M constants used in these inequalities will be explained below. Equalities (13) and (14) compute the total demand of a resource of all activities running during an event. Finally, Inequalities (15) ensure that all resource demands are met at all events.

Choosing the smallest possible Big-M constants for Inequalities (9)–(12) in DEF is important for making its LP relaxation as tight as possible. An easy way to set them is $M^{(9)}_{a,k} = T^{\max} - t^{\mathrm{r}}_a$, $M^{(10)}_{a,k} = t^{\mathrm{d}}_a - p_a - T^{\min}$, $M^{(11)}_{a,k} = T^{\max} - t^{\mathrm{r}}_a - p_a$, and $M^{(12)}_{a,k} = t^{\mathrm{d}}_a - T^{\min}$. However, by computing sets of activities that must precede or succeed a certain event in any feasible schedule, respectively, it is possible to fix some constants to zero. For details, we refer to Jatschka [2017].

The formulation has $O(|A'|^2)$ variables and $O(|R| \cdot |A'|^2)$ constraints. Thus, DEF is a compact model, but its LP relaxation typically yields rather weak LP bounds, primarily due to the inequalities involving the Big-M constants.

4.2 Time-indexed Formulation

In a classical MILP way, we can model SI-PTPSP by the following time-indexed formulation (TIF) using binary variables $x_{a,t}$ for indicating whether an activity $a \in A$ starts at time $t \in T_a$.

$$\min MS \tag{19}$$

$$\sum_{t \in T_a} x_{a,t} = 1 \qquad \forall a \in A \tag{20}$$

$$\sum_{t \in T_a} t \cdot x_{a,t} + p_a \leq MS \qquad \forall a \in A \tag{21}$$

$$\sum_{a \in A : r \in Q_a} \; \sum_{t' \in T_a : t \in Y_a(t')} x_{a,t'} \leq 1 \qquad \forall r \in R,\ t \in W_r \tag{22}$$

$$\sum_{t \in T_{a'}} t x_{a',t} - \sum_{t \in T_a} t x_{a,t} \geq p_a + L^{\min}_{a,a'} \qquad \forall (a,a') \in P \tag{23}$$

$$\sum_{t \in T_{a'}} t x_{a',t} - \sum_{t \in T_a} t x_{a,t} \leq p_a + L^{\max}_{a,a'} \qquad \forall (a,a') \in P \tag{24}$$

$$x_{a,t} \in \{0,1\} \qquad \forall a \in A,\ t \in T_a \tag{25}$$

$$MS \geq 0 \tag{26}$$

Equations (20) ensure that exactly one starting time is chosen for each activity. Inequalities (21) are used to determine the makespan MS. Resource restrictions are enforced by Inequalities (22). Last but not least, Constraints (23) and (24) guarantee that the precedence relations with their minimum and maximum time lags are respected.

The model has $O(|A| \cdot |T|)$ variables and $O(|T| \cdot (|A| + |R| + |P|))$ constraints. Typically, the LP relaxation of TIF yields substantially tighter dual bounds than the LP relaxation of DEF but its size and solvability strongly depend on the used time discretization, i.e., $|T|$.

5 Time-Bucket Relaxation

As the number of variables and constraints of TIF becomes fairly large when considering a fine-grained time discretization, directly solving the model may not be a viable approach in practice. We therefore consider a relaxation of it, in which we combine subsequent time slots into so-called *time-buckets*. This model, which we call time-bucket relaxation (TBR), yields a dual bound to the optimal value of the original problem but in general not a valid solution. Based on TBR we will build our iterative refinement approach in Section 6.

Let $B = \{B_1, \ldots, B_\beta\}$ be a partitioning of T into subsequent time-buckets. Note that the individual buckets do not need to have the same size. We denote by $I(B) = \{1, \ldots, \beta\}$ the index set of B. For all $b \in I(B)$ we define the set of consecutive time slots $B_b = \{B^{\text{start}}_b, \ldots, B^{\text{end}}_b\}$ contained in the bucket. Since B is a chronologically

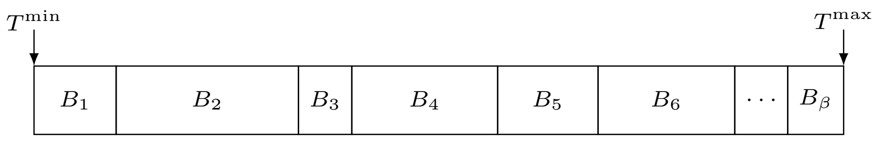


Figure 1: Bucket partitioning of T.

ordered partitioning of T, we have $B_1^{\text{start}} = T^{\min}$, $B_\beta^{\text{end}} = T^{\max}$, and $B_b^{\text{end}} + 1 = B_{b+1}^{\text{start}}$, $\forall b \in I(B) \setminus \{\beta\}$. For an illustration see Fig. 1. Additionally, let $W_r^B(b) = |B_b \cap W_r|$ denote the aggregated amount of resource $r \in R$ available over the whole bucket $b \in I(B)$.

Considering a bucket partitioning we now derive for each activity $a \in A$ all subsets of buckets in which the activity can be completely performed s.t. it executes at least partially in every bucket. We call these subsets *bucket sequences* of activity a and denote them by $C_a = \{C_{a,1}, \ldots, C_{a,\gamma_a}\} \subseteq 2^{I(B)}$. Let functions $\text{bfirst}(a,c)$ and $\text{blast}(a,c)$ for $a \in A$ and $c = 1, \ldots, \gamma_a$ provide the index of the first and the last bucket of bucket sequence $C_{a,c}$, respectively. The bucket sequences in C_a are assumed to be ordered according to increasing starting time, or, more precisely, lexicographically ordered according to $(\text{bfirst}(a,c), \text{blast}(a,c))$. We can determine all bucket sequences for an activity in time $O(|B| \log |B|)$, for details see Section 7.2. Analogous to set T_a we do not consider bucket sequences that involve only infeasible starting times.

For each bucket sequence let $S_{a,c}^{\min} \in T$ be the earliest time slot at which activity a can feasibly start when it is assigned to bucket sequence $C_{a,c} \in C_a$. Similarly, let $S_{a,c}^{\max} \in T$ be the latest possible starting point. Moreover, values $z_{a,b,c}^{\min}$ and $z_{a,b,c}^{\max}$ provide bounds on the number of utilized time slots within bucket $b \in C_{a,c}$ when activity a uses bucket-sequence $C_{a,c} \in C_a$. Note that for inner buckets b with $\text{bfirst}(a,c) < b < \text{blast}(a,c)$ we always have $z_{a,b,c}^{\min} = z_{a,b,c}^{\max} = |B_b|$. Fig. 2 shows a set of bucket sequences for a given activity. Observe that for bucket sequence $C_{a,2}$ we need to shift the execution window s.t. the activity executes at least for one time slot in bucket B_3, i.e., we require $z_{a,3,2}^{\min} > 0$ to avoid an overlap with bucket sequence $C_{a,1}$.

Our relaxation of TIF uses binary variables $y_{a,c}$ indicating whether activity $a \in A$ is performed in bucket sequence $C_{a,c}$ for $c \in \{1, \ldots, \gamma_a\}$. Model TBR is stated as follows:

$$\min MS \tag{27}$$

$$\sum_{c=1}^{\gamma_a} y_{a,c} = 1 \qquad \forall a \in A \tag{28}$$

$$\sum_{c=1}^{\gamma_a} S_{a,c}^{\min} \cdot y_{a,c} + p_a \le MS \qquad \forall a \in A \tag{29}$$

$$\sum_{a \in A : r \in Q_a} \sum_{C_{a,c} \in C_a : b \in C_{a,c}} z_{a,b,c}^{\min} \cdot y_{a,c} \le W_r^B(b) \qquad \forall r \in R,\ b \in I(B) \tag{30}$$

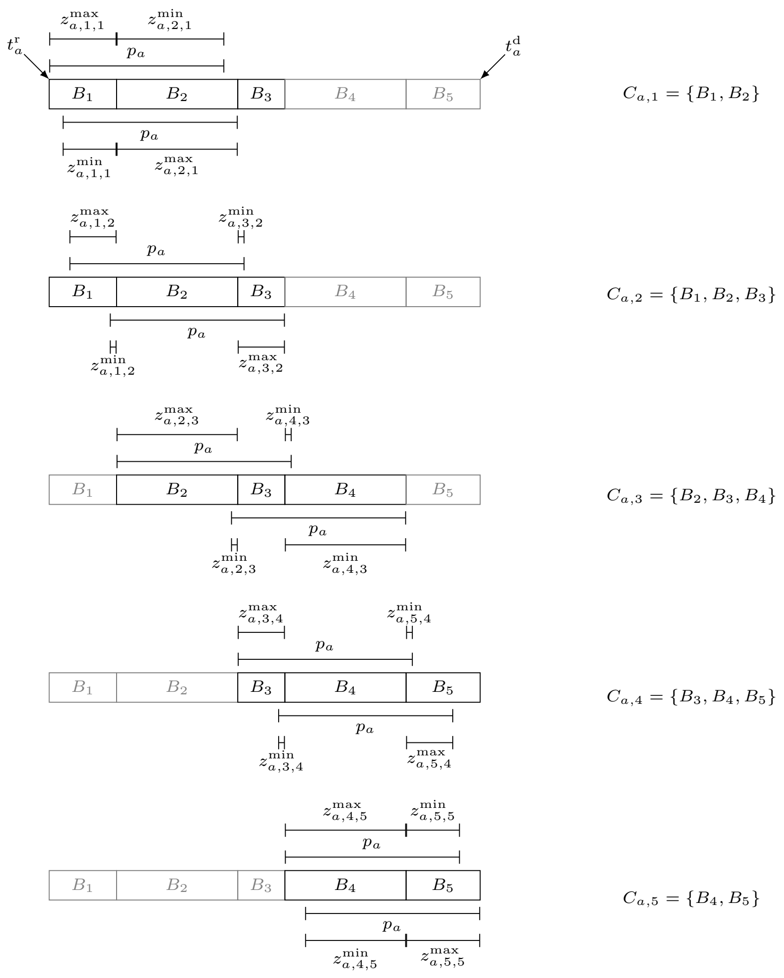


Figure 2: Bucket sequences C_a of an activity a with processing time p_a. Descriptions of inner buckets of a sequence are omitted since $z^{\min}_{a,b,c} = z^{\max}_{a,b,c} = |B_b|$ holds for them.

$$\sum_{c'=1}^{\gamma_{a'}} S^{\max}_{a',c'} \cdot y_{a',c'} - \sum_{c=1}^{\gamma_a} S^{\min}_{a,c} \cdot y_{a,c} \geq p_a + L^{\min}_{a,a'} \qquad \forall (a,a') \in P \tag{31}$$

$$\sum_{c'=1}^{\gamma_{a'}} S^{\min}_{a',c'} \cdot y_{a',c'} - \sum_{c=1}^{\gamma_a} S^{\max}_{a,c} \cdot y_{a,c} \leq p_a + L^{\max}_{a,a'} \qquad \forall (a,a') \in P \tag{32}$$

$$y_{a,c} \in \{0,1\} \qquad \forall a \in A, \; c = 1,\dots,\gamma_a \tag{33}$$

$$MS \geq 0 \tag{34}$$

Equations (28) ensure that exactly one bucket sequence is chosen for each activity. The makespan MS is determined using Inequalities (29). Constraints (30) consider the resource availabilities individually for each bucket in an accumulated fashion. Determined resource consumptions of activities are precise for all used inner buckets of a sequence but might underestimate the actually required amount in the first and last bucket. Finally, Inequalities (31) and (32) realize the precedence constraints with their minimum and maximum time lags, respectively. These restrictions also constitute a relaxation of the corresponding ones in TIF since the precise starting times within the buckets are not known (unless dealing with buckets of unit size).

The model has $O(|A| \cdot |B|)$ variables and $O(|A| + |R| \cdot |B| + |P|)$ constraints, and thus its size does not depend on $|T|$.

5.1 Comparison of TIF and TBR

First, let us consider the case of TBR in which all buckets have unit size, i.e., $B = \{\{T^{\min}\}, \{T^{\min}+1\}, \dots, \{T^{\max}\}\}$. Let us denote this special case by TBR$_1$. This leads to several simplifications. All buckets b belonging to some sequence $C_{a,c}$ are fully used, i.e., $z^{\min}_{a,b,c} = z^{\max}_{a,b,c} = |B_b| = 1$. Moreover, minimum and maximum starting times are equal and equivalent to the first time slot of the initial bucket of the sequence: $S^{\min}_{a,c} = S^{\max}_{a,c} = B^{\text{start}}_{\text{bfirst}(a,c)}$. Essentially, this means that $T_a = \{S^{\min}_{a,c} \mid C_{a,c} \in C_a\} = \{S^{\max}_{a,c} \mid C_{a,c} \in C_a\}$ and $|T_a| = |C_a|$ for all $a \in A$. Furthermore, since buckets correspond to original time slots in this scenario, resource availabilities become binary for each bucket.

For TIF and TBR$_1$ we consider function $\varphi_a \colon \{1, \dots \gamma_a\} \to T_a$ for each activity $a \in A$ with $\varphi_a(c) \coloneqq S^{\min}_{a,c}$.

Proposition 1. *Function φ is bijective.*

Proof. Each bucket sequence w.r.t. TBR$_1$ corresponds to a specific starting time. For each activity C_a considers all feasible bucket sequences and T_a all feasible starting times. Thus, there exists a unique mapping between these sets. □

Proposition 2. *The polyhedra of TBR$_1$ and TIF are isomorphic.*

Proof. We establish an isomorphism between the variables of the models using function φ_a and its inverse: $x_{a,t} = y_{a,\varphi_a^{-1}(t)}$ and $y_{a,c} = x_{a,\varphi_a(c)}$. Moreover, we can use these functions to immediately transform (20) into (28), (21) into (29), (23) into (31), and (24) into (32) and vice versa. To provide the isomorphism between (22) and (30) we need a few further things. First, recall that all $z^{\min}_{a,b,c}$ constants are equal to 1. Second, using $t \leftrightarrow \{t\}$ as isomorphism between T and the set of unit buckets we obtain $W^B_r(b) = 1$ if the corresponding time point $t \in W_r$ and $W^B_r(b) = 0$ otherwise. Finally, the correspondence between time points and unit buckets guarantees that $Y_a(t)$ and $C_{a,c}$ are isomorphic for $\varphi_a^{-1}(t) = c$. Putting things together also the resource constraints can be transformed into one another. □

Corollary 1. The LP relaxations of TBR$_1$ and TIF are equally strong.

Definition 1. Let TBR_B and $TBR_{B'}$ be two TBR-models with bucket partitionings B and B', respectively. $TBR_{B'}$ is called a refined model of TBR_B iff $\forall b' \in I(B')\ \exists b \in I(B)\ (B'_{b'} \subseteq B_b)$.

In the following we show that TBR_B is a relaxation of $TBR_{B'}$ and thus of TIF.

Definition 2. Let TBR_B be a TBR-model and let $TBR_{B'}$ be a refined model of TBR_B. Then, $\sigma\colon C'_a \to C_a$ defines a (surjective) mapping from bucket sequences C'_a w.r.t. $TBR_{B'}$ to bucket sequences C_a w.r.t. TBR_B satisfying for all $C'_{a,c'} \in C'_a$:

$$\bigcup_{b' \in C'_{a,c'}} b' \subseteq \bigcup_{b \in \sigma(C'_{a,c'})} b \wedge \forall C_{a,\hat{c}} \in C_a \left(\bigcup_{b' \in C'_{a,c'}} b' \not\subseteq \bigcup_{b \in C_{a,\hat{c}}} b \vee \sigma(C'_{a,c'}) \subseteq C_{a,\hat{c}} \right).$$

This means σ provides the inclusion minimal bucket sequence from TBR_B that contains at least the time slots that the bucket sequence from $TBR_{B'}$ contains.

Lemma 1. *Function σ can be implemented by:*

$$\sigma(C'_{a,c'}) = C_{a,c} \ \textit{s.t.}\ C_{a,c} \in C_a \wedge S^{\min}_{a,c'} \in \mathrm{bfirst}(a,c) \wedge (S^{\min}_{a,c'} + p_a) \in \mathrm{blast}(a,c)$$

Proof. Feasibility of $C'_{a,c'}$ together with the fact that buckets in $TBR_{B'}$ are subsets of those in TBR_B implies that there exists a sequence $C_{a,c} \in C_a$ satisfying $S^{\min}_{a,c'} \in \mathrm{bfirst}(a,c)$ and $(S^{\min}_{a,c'} + p_a) \in \mathrm{blast}(a,c)$. The buckets of sequence $C'_{a,c'}$ are contained in those of sequence $C_{a,c}$, i.e., $\bigcup_{b' \in C'_{a,c'}} b' \subseteq \bigcup_{b \in C_{a,c}} b$. Moreover, $C_{a,c}$ is uniquely determined since by definition two different bucket sequences cannot have the same first and last buckets. Therefore, every other sequence covering the buckets from $C'_{a,c'}$ must be strictly larger than $C_{a,c}$. □

Theorem 1. *Let TBR_B be a TBR-model and let $TBR_{B'}$ be a refined model of TBR_B. Then, TBR_B is a relaxation of $TBR_{B'}$.*

Proof. Using function σ according to Lemma 1, we create a solution y to TBR_B from an optimal solution y^* to $TBR_{B'}$ as follows:

$$y_{a,c} = \sum_{C'_{a,c'} \in C'_a : \sigma(C'_{a,c'}) = C_{a,c}} y^*_{a,c'} \qquad \forall a \in A, c \in \{1, \dots, \gamma_a\}$$

We first show that y is a feasible solution for TBR_B. Constraints (28) are satisfied since $y^*_{a,c'}$ is feasible and σ is surjective. As $\text{bfirst}(a,c') \subseteq \text{bfirst}(a,c)$ for all $C_{a,c} = \sigma(C'_{a,c'})$ it holds that $S^{\min}_{a,c} \leq S^{\min}_{a,c'}$ and $S^{\max}_{a,c'} \leq S^{\max}_{a,c}$. Hence, Constraints (29), (31), and (32) must hold. If Inequalities (30) are satisfied for y^*, then the resource constraints are also satisfied for y since the refined resource allocation entails the coarser one. Therefore, y is a feasible solution to TBR_B.

Since $S^{\min}_{a,c} \leq S^{\min}_{a,c'}$, the objective can only decline due to the transformation. Thus, the optimal solution value to TBR_B can be at most as large as the value of the optimal solution to $TBR_{B'}$. Thus, TBR_B is a relaxation of $TBR_{B'}$. □

Corollary 2. TBR is a relaxation of TIF.

5.2 Strengthening TBR by Valid Inequalities

In the following we introduce two types of valid inequalities to compensate for the loss of accuracy in TBR due to the bucket aggregation. Note that these inequalities strengthen the relaxation in general but might become redundant for more fine-grained bucket partitionings.

5.2.1 Clique Inequalities

Observe that two activities, represented by non-unit bucket sequences, cannot feasibly start in the same bucket if both require a certain resource. The same holds for two or more bucket sequences with these properties ending in the same bucket. This can be used to derive sets of incompatible bucket sequences that give rise to clique inequalities, see Demassey et al. [2005], Hardin et al. [2008].

To formulate respective constraints we determine for each $b \in I(B)$ sets $\mathcal{S}_b = \{(a,c) \mid a \in A, c \in C_a, z^{\min}_{a,b,c} < |B_b|, |C_{a,c}| > 1, \text{bfirst}(a,c) = b\}$ and $\mathcal{F}_b = \{(a,c) \mid a \in A, c \in C_a, z^{\min}_{a,b,c} < |B_b|, |C_{a,c}| > 1, \text{blast}(a,c) = b\}$ of non-unit bucket sequences starting and ending in bucket B_b, respectively. From each of these sets we derive a graph having the respective set as nodes and an edge between two nodes if the activities of the corresponding bucket sequences share a resource. Let $\mathcal{C}^{\mathcal{S}}_b$ and $\mathcal{C}^{\mathcal{F}}_b$ be the sets of all maximal cliques with a minimum size of two within these graphs. This leads to the following inequalities:

$$\sum_{(a,c)\in\kappa} y_{a,c} \leq 1, \qquad \forall b \in I(B), \forall \kappa \in \mathcal{C}^{\mathcal{S}}_b \tag{35}$$

$$\sum_{(a,c)\in\kappa} y_{a,c} \leq 1, \qquad \forall b \in I(B), \forall \kappa \in \mathcal{C}^{\mathcal{F}}_b \tag{36}$$

Some of these constraints might be redundant if the sum of $z^{\min}_{a,b,c}$ of the smallest two sequences is already large enough to prohibit them from being in the same bucket by means of Inequalities (30). The most trivial form of this case is excluded in the above sets by the condition $z^{\min}_{a,b,c} < |B_b|$.

The considered cliques can be computed using the algorithm by Bron and Kerbosch [1973]. Cazals and Karande [2008] show that this algorithm is worst-case optimal, i.e., it runs in $O(3^{\frac{n}{3}})$ which is the largest possible number of maximal cliques in a graph on n nodes. Although problematic in general, this might still be reasonable considering the rather small expected size of the conflict graphs.

Nevertheless, in our implementation we decided to avoid clique computations and resort to a simpler variant. We do so by considering a separate graph per resource obtaining a set of not necessarily maximal cliques. This leads to conceptually weaker inequalities but requires almost no computational overhead. More specifically, we consider subsets $\mathcal{S}_{b,r} = \mathcal{S}_b \cap \{(a, c) \mid a \in A, c \in C_a, r \in Q_a\}$ of $\mathcal{S}_b$ and subsets $\mathcal{F}_{b,r} = \mathcal{F}_b \cap \{(a, c) \mid a \in A, c \in C_a, r \in Q_a\}$ of $\mathcal{F}_b$, respectively, for $b \in I(B)$ and $r \in R$ s.t. within these subsets all activities require a common resource. Using these sets we formulate the following constraints:

$$\sum_{(a,c)\in\mathcal{S}_{b,r}} y_{a,c} \leq 1, \qquad \forall b \in I(B), \forall r \in R : |\mathcal{S}_{b,r}| \geq 2 \tag{37}$$

$$\sum_{(a,c)\in\mathcal{F}_{b,r}} y_{a,c} \leq 1, \qquad \forall b \in I(B), \forall r \in R : |\mathcal{F}_{b,r}| \geq 2 \tag{38}$$

If mutual overlap of the resources required by the activities is rare, the simpler inequalities are often almost as powerful as the full clique inequalities.

5.2.2 Path Inequalities

The idea of this kind of inequalities is to extend the precedence constraints (31) and (32) and the makespan constraints (34) to be valid for paths in the precedence graph instead of only for adjacent activities.

We consider the acyclic directed precedence graph $G = (A, P)$. Let $\pi_{a_0,a_m} = (a_0, a_1, \ldots, a_m)$ be a directed path from activity a_0 to activity a_m in G. Moreover, let minimum and maximum path lengths $d_{L^{\min}}(\pi_{a_0,a_m}) = \sum_{i=0}^{m-1} p_{a_i} + L^{\min}_{a_i,a_{i+1}}$ and $d_{L^{\max}}(\pi_{a_0,a_m}) = \sum_{i=0}^{m-1} p_{a_i} + L^{\max}_{a_i,a_{i+1}}$ be the minimum and maximum makespan of the activities within the path, respectively. Let $\Pi_{a,a'}$ denote the set of all distinct paths from node a to node a'. Since G is acyclic, $\Pi_{a,a'}$ is finite (but in general exponential in the number of edges) for all pairs of nodes $(a, a') \in A \times A : a \neq a'$. Let $\Pi = \bigcup_{a,a' \in A: a \neq a} \Pi_{a,a'}$ denote the union of all these paths between any two different nodes.

Let S be a feasible solution to SI-PTPSP. Then, for each path $\pi_{a,a'}$ in G it must hold that $S_a + d_{L^{\min}}(\pi_{a,a'}) \leq S_{a'}$ and $S_a + d_{L^{\max}}(\pi_{a,a'}) \geq S_{a'}$. Hence, adding the following inequalities for all $\pi_{a,a'} \in \Pi$ to TBR yields a strengthened relaxation of TIF:

$$\sum_{c=1}^{\gamma_a} S^{\min}_{a,c} \cdot y_{a,c} + d_{L^{\min}}(\pi_{a,a'}) \leq \sum_{c'=0}^{\gamma_{a'}-1} S^{\max}_{a',c'} \cdot y_{a',c'} \tag{39}$$

$$\sum_{c=1}^{\gamma_a} S^{\max}_{a,c} \cdot y_{a,c} + d_{L^{\max}}(\pi_{a,a'}) \geq \sum_{c'=0}^{\gamma_{a'}-1} S^{\min}_{a',c'} \cdot y_{a',c'} \tag{40}$$

$$\sum_{c=1}^{\gamma_a} S^{\min}_{a,c} \cdot y_{a,c} + d_{L^{\min}}(\pi_{a,a'}) + p_{a'} \leq MS \tag{41}$$

Due to the exponential number of these inequalities we only consider a reasonable subset of them in our implementation, for details see Section 7.3.

6 Iterative Time-Bucket Refinement Algorithm

For the original SI-PTPSP, TBR on its own is a method yielding a lower bound but no concrete feasible solution. The basic idea of the iterative time-bucket refinement algorithm (ITBRA) is to solve TBR repeatedly, refining the bucket partitioning in each iteration, until a proven optimal solution can be derived via primal heuristics or some other termination criterion is met. We will show that, given enough time, this algorithm converges to an optimal SI-PTPSP solution.

More specifically, we start by solving TBR with an initial bucket partitioning. Then, we try to heuristically derive a feasible SI-PTPSP solution that matches the objective value of TBR with a so-called gap closing heuristic (GCH). This heuristic fixes concrete times for activities in accordance with the TBR solution and guarantees to never violate resource or precedence constraints. If all activities can be scheduled in this way, we have found and optimal solution and the algorithm terminates. Otherwise, some activities remain unscheduled and we apply a follow-up heuristic to augment and repair the partial solution, possibly obtaining a feasible approximate solution and a primal bound. Here it can again be the case that we are able to close the optimality gap. If no provably optimal solution has been found thus far, we refine the bucket partitioning by splitting selected buckets and solve TBR again. For selecting the buckets to be refined and doing the splitting, we exploit information obtained from the TBR solution and the applied primal heuristics. This process is iterated until specified termination criteria are met or an optimal solution is found. The whole procedure is shown in Algorithm 1. The individual components of this approach will be explained in detail in the next sections.

6.1 Initial Bucket Partitioning

We create the initial bucket partitioning B in such a way that buckets start/end at any time when a resource availability interval starts or ends and at any release time and deadline of the activities. For details see Algorithm 2.

6.2 Primal Heuristics

We consider heuristics that attempt to derive feasible SI-PTPSP solutions and corresponding primal bounds based on TBR solutions. If ITBRA is terminated early, the best solution found in this way is returned. Note, however, that depending on the instance properties, these heuristics might also fail and then yield no feasible solution.

Algorithm 1: Iterative time-bucket refinement algorithm (ITBRA)

```
    Input: SI-PTPSP instance
    Output: solution to SI-PTPSP and lower bound
 1: compute initial bucket partitioning;
 2: compute initial primal solution;
 3: do
 4:     solve TBR for the current bucket partitioning;
 5:     apply gap closing heuristic (GCH): try to find an SI-PTPSP solution in
          accordance with the TBR solution;
 6:     if unscheduled activities remain then
 7:         apply follow-up heuristic to find feasible SI-PTPSP solution
 8:     end if
 9:     if gap closed then
10:         return optimal solution
11:     end if
12:     derive refined bucket partitioning for the next iteration;
13: while termination criteria not met;
14: return best heuristic solution and lower bound from TBR
```

6.2.1 Gap Closing Heuristic (GCH)

This is the first heuristic applied during an iteration of ITBRA. It attempts to construct an optimal solution according to TBR's result to close the optimality gap. Thus, it may only fully succeed when the relaxation's objective value does not underestimate the optimal SI-PTPSP solution value. If the gap cannot be closed, GCH provides only a partial solution and no primal bound. Information on the unscheduled activities then forms an important basis for the subsequent bucket refinement.

Let (y^*, MS^*) be the current optimal TBR solution. Initially, GCH receives for each activity $a \in A$ the interval $S_a^{\text{TBR}} = \{S_a^{\text{TBR,min}}, \ldots, S_a^{\text{TBR,max}}\}$ of potential starting times,

Algorithm 2: Computing an initial bucket partitioning

```
   Output: the initial bucket partitioning
1: $B \leftarrow \emptyset$; // bucket partitioning
2: $\mathcal{T} \leftarrow \{T^{\min}\} \cup \{T^{\max}+1\}$; // bucket starting times
3: $\mathcal{T} \leftarrow \mathcal{T} \cup \{W_{r,w}^{\text{start}}, W_{r,w}^{\text{end}}+1 \mid r \in R,\ w = 1,\ldots,\omega_r\}$;
4: $\mathcal{T} \leftarrow \mathcal{T} \cup \{t_a^{\text{r}}, t_a^{\text{d}} \mid a \in A\}$;
5: sort $\mathcal{T}$;
6: for $i \leftarrow 1$ to $|\mathcal{T}|-1$ do
7:     $B \leftarrow B \cup \{\{\mathcal{T}[i], \ldots, \mathcal{T}[i+1]-1\}\}$; // add bucket
8: end for
9: return $B$;
```

where $S_a^{\text{TBR,min}} = \sum_{c=0}^{\gamma_a - 1} S_{a,c}^{\min} \cdot y_{a,c}^*$ and $S_a^{\text{TBR,max}} = \sum_{c=0}^{\gamma_a - 1} S_{a,c}^{\max} \cdot y_{a,c}^*$. These intervals can in general be further reduced by removing for each $a \in A$ all time slots $t \in S_a^{\text{TBR}}$ violating at least one of the following conditions in relation to the precedence constraints and the calculation of the makespan:

$$\exists t' \in S_{a'}^{\text{TBR}} \ (t + p_a + L_{a,a'}^{\min} \leq t' \leq t + p_a + L_{a,a'}^{\max}) \qquad \forall (a, a') \in P \tag{42}$$

$$\exists t' \in S_{a'}^{\text{TBR}} \ (t' + p_{a'} + L_{a',a}^{\min} \leq t \leq t' + p_{a'} + L_{a',a}^{\max}) \qquad \forall (a', a) \in P \tag{43}$$

$$t + p_a \leq MS^* \tag{44}$$

We prune the set of intervals of potential activity starting times for all activities $S^{\text{TBR}} = \{S_a^{\text{TBR}} \mid a \in A\}$ so that *arc consistency* is achieved w.r.t. Conditions (42)–(44). This is done by constraint propagation with a method like the well-known AC3 algorithm, see Mackworth [1977]. Note that this constraint propagation may yield empty intervals for some activities, indicating that there remains no feasible starting time assignment respecting all constraints. In this case GCH will give up on this activity and continues with the remaining ones deviating from the usual arc consistency concept to allow further activities to be scheduled.

The pseudocode of GCH is shown in Algorithm 3. After the initial pruning of starting time intervals, GCH constructs the (partial) schedule S by iteratively scheduling the activities respecting all constraints. If this is not possible for some activities, they remain unscheduled. Using a greedy strategy, the activities are considered in non-decreasing order of $S_a^{\text{TBR,max}} + p_a$, i.e., according to their earliest possible finishing times. Activities are always scheduled at the earliest feasible time from S_a^{TBR}. Note that any explicit enumeration of time slots from an interval can be efficiently avoided by using basic interval arithmetic. Whenever an activity starting time is set, constraint propagation is repeated to ensure arc consistency according to Conditions (42)–(44).

If GCH fails to close the gap, we attempt to compute a feasible solution instead that might have a larger objective value than the current TBR bound.

6.2.2 Activity Block Construction Heuristic (ABCH)

This algorithm is based on the idea of first constructing so-called *activity blocks*, which correspond to the weakly connected components of the precedence graph. All the activities belonging to one such weakly connected component are statically linked considering the precedence constraints and minimum time lags between them. ABCH then greedily schedules the activity blocks that have not been scheduled completely by GCH instead of the individual activities. The activity blocks are considered in order of their release times and are scheduled at the first time slot where no resource constraint is violated w.r.t. the activity block's individual activities and resource requirements. Details are provided in Algorithm 4.

6.2.3 Greedy Randomized Adaptive Search Procedure (GRASP)

GRASP is a prominent metaheuristic that applies a randomized variant of a construction heuristic followed by a local search component independently many times, where the

Algorithm 3: Gap closing heuristic (GCH)

```
    Input: intervals of potential starting times $S^{\mathrm{TBR}} = \{S_a^{\mathrm{TBR}} \mid a \in A\}$ with
           $S_a^{\mathrm{TBR}} = \{S_a^{\mathrm{TBR,min}}, \dots, S_a^{\mathrm{TBR,max}}\}$
    Output: (partial) schedule $S$ and all activities that cannot be scheduled w.r.t.
            $S^{\mathrm{TBR}}$ grouped by violation type
 1: $A_P \leftarrow \emptyset$; // activities with violated precedence constraints
 2: $A_R \leftarrow \emptyset$; // activities with violated resource constraints
 3: $A_U \leftarrow A$; // unscheduled activities
 4: $W'_r \leftarrow W_r$; // resource availabilities
 5: prune potential starting time intervals $S^{\mathrm{TBR}}$;
 6: while $A_U \neq \emptyset$ do
 7:     select and remove an activity $a \in A_U$ with minimal $S_a^{\mathrm{TBR,max}} + p_a$;
 8:     if $S_a^{\mathrm{TBR}} = \emptyset$ then // precedence constraints violated
 9:         $A_P \leftarrow A_P \cup \{a\}$;
10:         continue;
11:     end if
12:     $\overline{S_a^{\mathrm{TBR}}} \leftarrow \{t \in S_a^{\mathrm{TBR}} \mid \{t, \dots, t + p_a - 1\} \subseteq W'_r, \forall r \in Q_a\}$;
13:     if $\overline{S_a^{\mathrm{TBR}}} = \emptyset$ then // resource constraints violated
14:         $A_R \leftarrow A_R \cup \{a\}$;
15:         continue;
16:     end if
17:     $S_a \leftarrow \min\ \overline{S_a^{\mathrm{TBR}}}$;
18:     $S_a^{\mathrm{TBR}} \leftarrow \{S_a\}$;
19:     $W'_r \leftarrow W'_r \setminus \{t, \dots, t + p_a - 1\},\ \forall r \in Q_a$;
20:     prune potential starting time intervals $S^{\mathrm{TBR}}$;
21: end while
22: return $S, A_P, A_R$;
```

best found solution is kept as the result, see Resende and Ribeiro [2010]. We consider GRASP here as an advanced alternative to ABCH within ITBRA. The approach provides a reasonable balance between being still relatively simple but providing considerably better results than ABCH. There are clearly other options but our aim here is to keep standard metaheuristic aspects simple in order to put more emphasis on TBR's and ITBRA's fundamentals.

Both, GCH and ABCH can be randomized. We do so by allowing the order in which the activities or activity blocks are considered to deviate from the strict greedy criterion. In particular, we choose uniformly at random from the $k_{\mathrm{GCH}}^{\mathrm{grand}}$ ($k_{\mathrm{ABCH}}^{\mathrm{grand}}$) candidates with the highest priority. Parameters $k_{\mathrm{GCH}}^{\mathrm{grand}}$ and $k_{\mathrm{ABCH}}^{\mathrm{grand}}$ control the strength of the randomization. Note that the success of ABCH and hence also of the GRASP strongly depends on the partial solution provided by GCH. Therefore, we primarily choose to randomize GCH. We also try to compute a primal solution at the very beginning before solving

Algorithm 4: Activity block construction heuristic (ABCH)

```
    Input: a partial schedule $S^{\mathrm{GCH}}$ computed by GCH
    Output: a feasible schedule $S$ or no solution if $S^{\mathrm{GCH}}$ cannot be completed
 1: $C \leftarrow$ set of subsets of $A$ corresponding to the weakly connected components of
      the precedence graph which are not completely scheduled in $S^{\mathrm{GCH}}$;
 2: $A^C \leftarrow \emptyset$; // the set of activity blocks
 3: forall weakly connected components $c \in C$ do
 4:     $S^c \leftarrow \emptyset$ ; // a schedule representing the activity block of $c$
 5:     forall activities $a \in c$ in topological order do
 6:         schedule $a$ in $S^c$ at the earliest possible time w.r.t. the precedence
              constraints and resource consumptions of activities in $c$ but ignoring all
              other activities as well as release times and deadlines, and resource
              availabilities;
 7:     end
 8:     the release time of the activity block is $\min_{a \in c} t^{\mathrm{r}}_a$;
 9:     $A^C \leftarrow A^C \cup \{S^c\}$ ;
10: end
11: forall activity blocks $S^c \in A^C$ ordered according to release time do
12:     try to schedule the activity block at the earliest feasible time in $S$ s.t. activity
          release times and deadlines as well as resource constraints are satisfied;
13:     if no feasible time found then
14:         return no solution;
15:     end if
16: end
17: return $S$;
```

TBR for the first time. Hence, there is no GCH solution available at this point. In this case we randomize ABCH instead.

To get a strong guidance for the bucket refinement process we prefer GCH solutions that schedule as many activities as possible. However, these solutions might not necessarily correspond to those solutions that work best in conjunction with ABCH. Therefore, we track the best complete solution and the best partial GCH solution separately. This means that our GRASP returns a feasible SI-PTPSP solution as well as a partial GCH solution (which might be unrelated). Since GRASP combines the functionalities of GCH and ABCH, it effectively replaces Lines 5–8 in Algorithm 1.

We consider a local search component using a classical 2-exchange neighborhood on the order of the activity blocks scheduled by ABCH. The local search is always performed until a local optimum is reached.

As termination criterion for the GRASP a combination of a time limit and a maximal number of iterations without improvement is used, details will be given in Section 8. Moreover, in the first iteration of the GRASP the deterministic versions of GCH and ABCH are used. This guarantees, especially for short executions, that the final result of

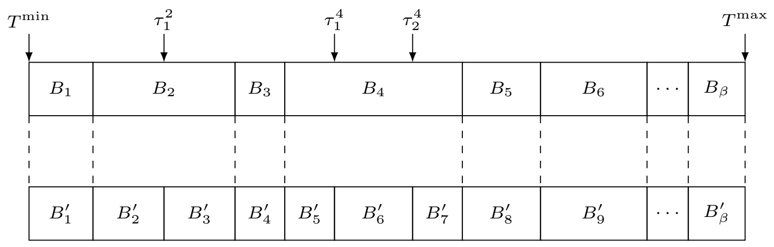


Figure 3: An example of a bucket refinement for $\tau^2 = \{\tau_1^2\}$, $\tau^4 = \{\tau_1^4, \tau_2^4\}$, and $\tau^b = \emptyset$ for $b \in I(B) \setminus \{2, 4\}$.

the GRASP is never worse than the one of the pure heuristics.

6.3 Bucket Refinement Strategies

In general, the bucket refinement is done by selecting one or more existing buckets and splitting each of them at selected points into two or more new buckets. If a bucket consists of only a single time slot, it cannot be subdivided further and becomes irrelevant for subsequent splitting decisions. Buckets are never merged or extended in our approach, i.e., the number of buckets always strictly increases due to the refinement. This guarantees that ITBRA eventually terminates if at least one bucket is subdivided in each iteration (cf. Theorem 1).

More formally, a refinement of some bucket $B_b \in B$ is given by an ordered set of splitting points $\tau^b = \{\tau_1^b, \ldots, \tau_m^b\} \subseteq \{B_b^{\text{start}}+1, \ldots, B_b^{\text{end}}\}$ with $\tau_1^b < \ldots < \tau_m^b$. Based on τ^b we get $|\tau^b|+1$ new buckets replacing the original one: $\{B_b^{\text{start}}, \ldots, \tau_1^b-1\}, \{\tau_1^b, \ldots, \tau_2^b-1\}, \ldots, \{\tau_m^b, \ldots, B_b^{\text{end}}\}$. For an example see Fig. 3.

In general, the decisions to be made in the bucket refinement process are (a) which buckets are to be refined, (b) at which positions, and (c) how many splits to apply. To address these tasks we need criteria that identify promising bucket refinements. Most importantly, a bucket refinement should affect the current optimal TBR solution in order to guarantee that not the same bucket sequences comprise an optimal solution again. In this way, it is ensured that we obtain a more refined solution in each iteration. Furthermore, bucket splitting should be done in such a way that it is beneficial for the heuristics, helping them to find good feasible solutions. Therefore, constraints that were responsible for leaving activities unscheduled in the heuristics should be exploited to prevent these situations from occurring again. Last but not least, we want to obtain a dual bound for the SI-PTPSP that is as tight as possible. Hence, a bucket refinement that likely has implications on TBR's objective value is desirable.

6.3.1 Selecting Buckets to Refine

Observe that refining inner buckets of selected bucket sequences does not directly affect the current TBR solution. Refining first and last buckets (if they are non-unit buckets), however, ensures that the bucket sequence that contained them does not exist in the refined TBR anymore and therefore cannot be used again. Furthermore, some newly introduced buckets might not be part of feasible bucket sequences anymore, resulting in a more restricted scenario. Hence, we want to either split only first or last buckets of selected sequences or both. If we use just one bucket, we need to resort to the other one if otherwise no progress can be made. During preliminary tests it turned out that always using both boundary buckets for refinement is superior. Another question is for which bucket sequences the bounding buckets shall be refined. In the following we propose different strategies that will be experimentally compared in Section 8.2.2.

All Selected (ASEL) Using this strategy we refine all first and last buckets of all bucket sequences selected in the current optimal TBR solution. This can, however, be inefficient as it may increase the total number of buckets in each iteration substantially. The following strategies will therefore only consider certain subsets.

All In GCH Schedule (AIGS) We refine all first and last buckets of only those bucket sequences whose corresponding activities could be feasibly scheduled by GCH. The idea is to improve accuracy for the scheduled activities in order to reveal sources of infeasibility w.r.t. the activities that could not be scheduled once TBR is solved the next time.

Violated Due (VDUE) If GCH fails to schedule all activities, it provides a set of activities A_P that cannot be scheduled due to the precedence constraints and a set of activities A_R that cannot be scheduled due to the resource constraints. The basic idea is to refine buckets related to activities in the schedule that immediately prevent the activities in A_P and A_R from being scheduled. To identify these activities we consider the partial schedule S generated by GCH.

Let $A^{\mathrm{GCH}} = A \setminus (A_R \cup A_P)$ be the set of feasibly scheduled activities. Refinements based on resource infeasibilities are derived from sets $N_R(a) = \{a' \in A^{\mathrm{GCH}} \mid Q_a \cap Q_{a'} \neq \emptyset \wedge \{S_{a'}, \ldots, S_{a'} + p_{a'} - 1\} \cap \{S_a^{\mathrm{TBR,min}}, \ldots, S_a^{\mathrm{TBR,max}} + p_a - 1\} \neq \emptyset\}$ for $a \in A_R$. For each activity $a' \in N_R(a)$ we refine the first and last bucket of the bucket sequence $C_{a',c}$ in the TBR solution.

The activities potentially responsible for $a \in A_P$ having no valid starting time are the activities a' in A^{GCH} s.t. $(a, a') \in P$ or $(a', a) \in P$. However, we do not have to consider all activities incident to a for the refinement. Let $N_P^-(a) = \{a' \mid (a', a) \in P \wedge a' \in A^{\mathrm{GCH}}\}$ and $N_P^+(a) = \{a' \mid (a, a') \in P \wedge a' \in A^{\mathrm{GCH}}\}$ for all $a \in A_P$. Then, calculate:

$$\begin{aligned} N_P(a) = & \mathop{\arg\max}_{a' \in N_P^-(a)} \{S_{a'} + p_{a'} + L^{\min}_{a',a}\} \quad \cup \quad \mathop{\arg\min}_{a' \in N_P^-(a)} \{S_{a'} + p_{a'} + L^{\max}_{a',a}\} \quad \cup \\ & \mathop{\arg\min}_{a' \in N_P^+(a)} \{S_{a'} - L^{\max}_{a,a'}\} \quad \cup \quad \mathop{\arg\max}_{a' \in N_P^+(a)} \{S_{a'} - L^{\min}_{a,a'}\} \end{aligned} \tag{45}$$

We refine the first and last buckets of all bucket sequences of activities $a' \in N_P(a)$ that are selected in the current TBR solution. If no refinement is possible for bucket sequences corresponding to $a' \in N_R(a) \cup N_P(a)$, we refine the first and last bucket of $C_{a,c}$ instead.

6.3.2 Identifying Splitting Positions

Once a bucket has been selected for refinement, we have to decide at which position(s) it shall be subdivided. Again, we consider different strategies. The challenge is to identify candidate positions that usually have a large impact on the subsequent TBR and its solution while resulting in well-balanced sub-buckets.

Binary (B) Let $C_{a,c}$ be the bucket sequence causing its first and last buckets to be selected for refinement. We split the selected buckets in such a way that the interval of potential starting and finishing times of the respective activity is bisected. In particular, for $\mathrm{bfirst}(a, c)$ and $\mathrm{blast}(a, c)$, we consider the splitting positions $\lceil (S^{\min}_{a,c} + S^{\max}_{a,c})/2 \rceil$ and $\lceil (S^{\max}_{a,c} + S^{\min}_{a,c})/2 \rceil + p_a$, respectively. We have to round up in case of non integral refinement positions since it is not feasible to refine w.r.t. the bucket start. Although this approach typically leads to well-balanced sub-buckets, it might often have a rather weak impact on the subsequent TBR solution because the resulting buckets might still be too large to reveal certain sources of infeasibility.

Start/End Time (SET) Let a be an activity that could be scheduled by GCH and $C_{a,c}$ the corresponding bucket sequence in TBR whose first and last buckets shall be refined. We split $\mathrm{bfirst}(a, c)$ at the activity's starting time S_a and $\mathrm{blast}(a, c)$ at $S_a + p_a$, i.e., after activity a has ended according to GCH's schedule. Thus, the specifically chosen time assignment of GCH gets an individual bucket sequence in the next iteration.

Because this method is defined only for activities that could be scheduled by GCH, it is applicable only in direct combination with AIGS. To overcome this limitation we resort to B if SET is not applicable. The obtained strategy is denoted by SET+B.

6.3.3 Selecting Splitting Positions

The strategies introduced above may yield several splitting positions for a single bucket, especially since the same bucket may be selected multiple times for refinement for different activities. In principle, we want to generate as few new buckets as possible while ensuring strong progress w.r.t. the dual bound and narrowing down the activities' possible starting time intervals. Splitting at all identified positions might therefore not be the best option. In the following we propose different strategies for selecting for each selected bucket the splitting positions to be actually used from all positions determined in the previous step. Let set τ^b be this union of identified splitting positions for bucket b.

Union Refinement (UR) We simply use all identified splitting positions. As already mentioned, however, this approach may lead to a high increase in the number of buckets and may therefore not be justified.

Binary Refinement (BR) We use the splitting position $\tau' \in \tau^b$ closest to the center of the bucket, i.e., $\tau' = \arg\min_{t \in \tau^b} \left| \frac{B^{\text{start}}_b + B^{\text{end}}_b}{2} - t \right|$; ties are broken according to the order

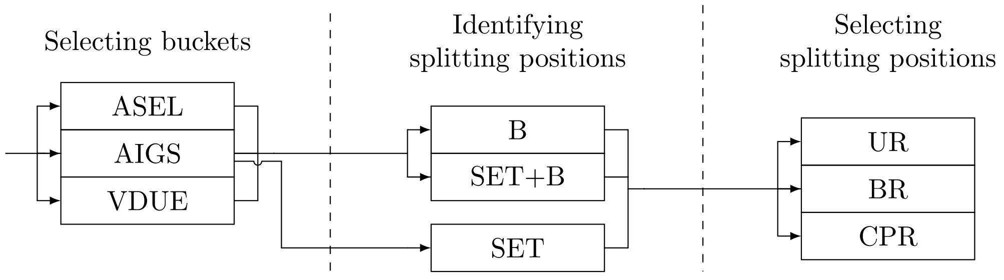

Selecting buckets
Identifying splitting positions
Selecting splitting positions
ASEL
AIGS
VDUE
B
SET+B
SET
UR
BR
CPR

Figure 4: Overview of the proposed strategies to perform a bucket refinement and how they can be combined.

in which the splitting positions have been obtained. This approach clearly tends to keep the number of buckets low but may increase the total number of required iterations of ITBRA.

Centered Partition Refinement (CPR) We first partition τ^b into two sets at $\bar{t} = \frac{B_b^{\text{start}}+B_b^{\text{end}}}{2}$. Let $\tau^{b,\text{l}} = \{t \in \tau^b \mid t \leq \bar{t}\}$ and $\tau^{b,\text{r}} = \{t \in \tau^b \mid t > \bar{t}\}$. To obtain up to three new buckets we choose as splitting points the two "innermost" elements, i.e., we apply the refinement $\{\max \tau^{b,\text{l}}, \min \tau^{b,\text{r}}\}$. If one of the sets is empty, we apply only a single split.

The idea of this partitioning is to give candidate positions close to either boundary of the bucket equal chances of being selected. Splitting a bucket close to its end usually has a strong influence on (non-unit) bucket sequences starting in the bucket while choosing a splitting position close to the start typically has a higher impact on (non-unit) bucket sequences ending in this bucket. Prioritizing splitting positions close to the center of the bucket results in a more balanced subdivision.

6.3.4 Further Considerations

We also investigated bucket selection techniques based on critical paths, see Guerriero and Talarico [2010]. This means that we consider sequences of activities that directly define the makespan. However, our experiments indicate that bucket refinements based on this strategy do not work well. We therefore omit them, as well as a few other inferior techniques, here and refer the interested reader to Jatschka [2017] for further details.

Fig. 4 provides an overview of the discussed bucket selection, splitting position identification, and splitting position selection strategies.

7 Implementation Details

In this section, we discuss further algorithmic details that are important for an efficient implementation of ITBRA and the associated heuristics.

7.1 Preprocessing Activity Starting Times

To obtain the restricted set of possible activity starting times T_a we start by discarding the starting times leading to resource infeasibilities:

$$T_a = \{t \in T \mid t^{\mathrm{r}}_a \le t \le t^{\mathrm{d}}_a - p_a, \forall r \in Q_a \; \forall t' \in Y_a(t) \; (t' \in W_r)\}$$

The obtained set is then further reduced by taking also precedence relations into account. In particular, only starting times respecting the following conditions are feasible:

$$\begin{aligned}
&\forall (a,a') \in P \; \exists t' \in T_{a'} \; (t + p_a + L^{\min}_{a,a'} \le t' \le t + p_a + L^{\max}_{a,a'}) \\
&\forall (a',a) \in P \; \exists t' \in T_{a'} \; (t' + p_{a'} + L^{\min}_{a',a} \le t \le t' + p_{a'} + L^{\max}_{a',a})
\end{aligned}$$

To achieve arc consistency w.r.t. them we can use constraint propagation similar as in GCH. All these calculations can be performed based on interval arithmetic without enumerating individual time slots, and thus in time independent of $|T|$.

Finally, the originally given release times and deadlines can be tightened according to the pruned sets T_a, i.e., we set

$$\begin{aligned}
t^{\mathrm{r}}_a &\leftarrow \min T_a & \forall a \in A \\
t^{\mathrm{d}}_a &\leftarrow p_a + \max T_a & \forall a \in A
\end{aligned}$$

7.2 Computing Bucket Sequences

Algorithm 5 calculates the bucket sequences C_a for an activity $a \in A$ using the fact that bucket sequences are uniquely determined by their earliest possible starting times $S^{\min}_{a,c}$. In particular, we can efficiently compute the next such time point that needs to be considered from the previous one.

If the current bucket sequence consists of a single bucket, we proceed with the time point ensuring that only $p_a - 1$ time can be spent in the current bucket, see Line 12. Otherwise, we try to find the earliest time point that guarantees that we start in b^{first} and finish in bucket $b^{\text{last}} + 1$. If no such time point exists, we proceed with the earliest time slot in bucket $b^{\text{first}} + 1$ instead or stop if the activity's deadline has already been reached. The offset, denoted by δ, to the sought time point can be computed according to Line 16.

Iterating over the earliest starting times is linear in the number of buckets. The bucket to which a certain time slot belongs can be determined in logarithmic time w.r.t. the number of buckets. Hence, the overall time required by the algorithm is in $O(|B| \log |B|)$. Note that the $z^{\min}_{a,b,c}$ and $z^{\max}_{a,b,c}$ values are only set for the first and last buckets of the computed sequences since these values are always equal to the bucket size for all inner buckets.

For $C_{a,c} \in C_a$ let $T^{\mathrm{s}}_{a,c} = \{S^{\min}_{a,c}, \ldots, S^{\max}_{a,c}\} \cap T_a$. We can discard all bucket sequences for which $T^{\mathrm{s}}_{a,c} = \emptyset$. Moreover, $S^{\min}_{a,c}$ and $S^{\max}_{a,c}$ can be tightened by setting $S^{\min}_{a,c}$ to $\min(T^{\mathrm{s}}_{a,c})$ and $S^{\max}_{a,c}$ to $\max(T^{\mathrm{s}}_{a,c})$.

Algorithm 5: Computing all bucket sequences for an activity.

```
    Input: Activity $a \in A$
    Output: Set of bucket sequences $C_a$, associated values $S^{\min}_{a,c}$, $S^{\max}_{a,c}$, $z^{\min}_{a,b,c}$, and $z^{\max}_{a,b,c}$
 1: $C_a \leftarrow \emptyset$;
 2: $t \leftarrow t^{\mathrm{r}}_a$;
 3: $c \leftarrow 1$;
 4: while $t \leq t^{\mathrm{d}}_a - p_a$ do
 5:     $b^{\text{first}} \leftarrow b : t \in B_b$;
 6:     $b^{\text{last}} \leftarrow b : (t + p_a - 1) \in B_b$;
 7:     $C_{a,c} \leftarrow \{B_{b^{\text{first}}}, \ldots, B_{b^{\text{last}}}\}$;
 8:     $S^{\min}_{a,c} \leftarrow t$;
 9:     if $b^{\text{first}} = b^{\text{last}}$ then
10:         $z^{\min}_{a,b^{\text{last}},c} \leftarrow p_a$;
11:         $z^{\max}_{a,b^{\text{last}},c} \leftarrow p_a$;
12:         $t \leftarrow B^{\text{end}}_{b^{\text{last}}} - p_a + 2$;
13:     else
14:         $z^{\max}_{a,b^{\text{first}},c} \leftarrow B^{\text{end}}_{b^{\text{first}}} - t + 1$;
15:         $z^{\min}_{a,b^{\text{last}},c} \leftarrow S^{\min}_{a,c} + p_a - B^{\text{start}}_{b^{\text{last}}}$;
16:         $\delta \leftarrow \min\left\{z^{\max}_{a,b^{\text{first}},c} - 1, \min\left\{B^{\text{end}}_{b^{\text{last}}}, t^{\mathrm{d}}_a - 1\right\} - \left(S^{\min}_{a,c} + p_a - 1\right)\right\}$;
17:         $z^{\min}_{a,b^{\text{first}},c} \leftarrow z^{\max}_{a,b^{\text{first}},c} - \delta$;
18:         $z^{\max}_{a,b^{\text{last}},c} \leftarrow z^{\min}_{a,b^{\text{last}},c} + \delta$;
19:         $t \leftarrow S^{\min}_{a,c} + \delta + 1$;
20:     end if
21:     $S^{\max}_{a,c} = B^{\text{end}}_{b^{\text{first}}} - z^{\min}_{a,b^{\text{first}},c} + 1$;
22:     $C_a \leftarrow C_a \cup \{C_{a,c}\}$;
23:     $c \leftarrow c + 1$;
24: end while
25: return $C_a$;
```

7.3 Valid Inequalities

As already mentioned, we only consider the simplified version of the clique inequalities (37) and (38) to avoid the overhead for computing maximal cliques. The number of these inequalities grows significantly as the buckets get more fine-grained. Fortunately, the final bucket partitionings turned out to be still sufficiently coarse to add all inequalities of this type to the initial formulation.

Recall that the number of path inequalities (39)–(41) is in general exponential. In favor of keeping the model compact we avoided dynamic separation and consider only a reasonable subset of these inequalities that is added in the beginning. Clearly, we want to use a subset of the paths Π still having a strong influence on the relaxation. The idea is to use all paths targeting nodes of the precedence graph with an out-degree of zero. This guarantees that precedence relations are enforced more strictly between all sinks and their predecessors. Since the sinks in the precedence graph are the nodes that will define the makespan, this appears to be particularly important.

To this end, we consider the following subsets of Π with $\deg^+(\cdot)$ denoting the out-degree of a node:

$$\Pi_{L^{\min}} = \bigcup_{a,a' \in A: a \neq a'} \{ \underset{\pi_{a,a'} \in \Pi_{a,a'}}{\arg\max}\ d_{L^{\min}}(\pi_{a,a'}) \mid \Pi_{a,a'} \neq \emptyset, \deg^+(a') = 0\}$$

$$\Pi_{L^{\max}} = \bigcup_{a,a' \in A: a \neq a'} \{ \underset{\pi_{a,a'} \in \Pi_{a,a'}}{\arg\min}\ d_{L^{\max}}(\pi_{a,a'}) \mid \Pi_{a,a'} \neq \emptyset, \deg^+(a') = 0\}$$

We then add Inequalities (39) and (41) only for paths $\pi_{a,a'} \in \Pi_{L^{\min}}$ and Inequalities (40) only for paths $\pi_{a,a'} \in \Pi_{L^{\max}}$.

8 Computational Study

In this section we are going to present the computational results for the considered algorithms with their variants. We start by giving details on the used test instances and the motivation for their selection. Then, we provide details on the actually used configurations. Finally, we present the obtained results.

8.1 Test Instances

The benchmark instances are motivated by the real world patient scheduling scenario at cancer treatment center MedAustron that requires scheduling of particle therapies. In general, each treatment session consists of five activities that have to be performed sequentially. The modeled resources are the particle beam, the irradiation rooms, the radio oncologists, and the anesthetist. In principle, resources are assumed to be available for the whole time horizon except for short time periods. The most critical resource is the particle beam, which is required by exactly one activity of each treatment. The particle beam is shared between three irradiation rooms, in which also additional preparation and follow-up tasks have to be performed. A radio oncologist is required for the first and the last activity, respectively. In addition, some patients require sedation, which means that the anesthetist is involved in all activities.

The main characteristic of our benchmark instances is the number of activities. We have generated two groups of benchmark instances, each consisting of 15 instances per number of activities $\alpha \in \{20, 30, \dots, 100\}$. These two groups differ in the size of the interval between release time and deadline of the activities and with it their difficulty.

Activities are generated treatment-wise, i.e., by considering sequences of five activities at a time. The particle beam resource is required by the middle activity, i.e., the third one. The second, third, and fourth activity demand one of the room resources selected uniformly at random. We assume that $\lceil \frac{\alpha}{10} \rceil$ radio oncologists are available and select one of them for the first and last activity. Moreover, 25% of the treatments are assumed to require sedation and are therefore associated with the anesthetist resource. We add for each consecutive activity in the treatment sequence a minimum and maximum time lag. Hence, the resulting precedence graph consists of connected components, each being a path of length five. In the following we refer to these paths, that essentially are equivalent to the treatments, also as *chains*. The processing times of the activities are randomly chosen from the set $\{100, \dots, 10000\}$. Minimum lags are always 100 and maximum lags are always 10000.

It remains to set the release times and deadlines of the activities and the resources' availability windows in such a way that the resulting benchmark instances are feasible with high probability but not trivial. For this reason a preliminary naïve schedule is generated from which release times and deadlines are derived. To this end, the activities are placed treatment-wise in the tentative time horizon $\{0, \dots, \sum_{a \in A}(p_a + 10000)\}$ by randomly selecting a starting time for the first activity of each connected component. For the subsequent activities a random time lag in $\{L^{\min}_{a,a'}, \dots, L^{\max}_{a,a'}\}$ is enforced. If a determined starting time of an activity conflicts with an already scheduled one, the connected component is reconsidered.

From this preliminary schedule we derive tentative release times and deadlines which are then scaled to receive a challenging instance. We consider two variants to generate a group of "easy" and a group of "hard" instances. The latter features larger release time deadline windows that make the respective instances more challenging. For details on the used scaling factors see Jatschka [2017].

Finally, the availability of the resources is restricted. Each resource has five to seven time windows during which it is unavailable. The duration of these time windows is randomly chosen from the set $\{700, \dots, 1500\}$. The positions of these unavailability windows are chosen uniformly at random from the set $\{0, \dots, T^{\max}\}$.

To our best knowledge benchmark instances considering a comparable scenario do not exist. Our newly introduced test instances are made available at http://www.ac.tuwien.ac.at/research/problem-instances. An overview of the basic characteristics of the test instances is provided in Table 1. Instance sets are named according to $[e|h]_\alpha$ where e stands for the "easy" group of instances and h for the "hard" ones, and α indicates the considered number of activities. Each instance set consists of 15 instances.

Table 1: Characteristics of the test instances grouped by difficulty and number of activities. The subscripts indicate the number of activities per instance. $T^{\max}$ denotes the average scheduling horizon. The number of resources ρ and the number of chains (chains) is the same for all instance of a set.

set	$T^{\max}$	ρ	chains	set	$T^{\max}$	ρ	chains
e_{20}	104 649	7	4	h_{20}	104 575	7	4
e_{30}	138 808	8	6	h_{30}	137 745	8	6
e_{40}	169 642	9	8	h_{40}	167 003	9	8
e_{50}	198 386	10	10	h_{50}	201 269	10	10
e_{60}	220 792	11	12	h_{60}	220 606	11	12
e_{70}	244 279	12	14	h_{70}	244 788	12	14
e_{80}	271 461	13	16	h_{80}	271 327	13	16
e_{90}	293 110	14	18	h_{90}	289 278	14	18
e_{100}	316 316	15	20	h_{100}	317 324	15	20

8.2 Computational Experiments

The test runs have been executed on an Intel Xeon E5540 with 2.53 GHz using a time limit of 7200 seconds and a memory limit of 4GB RAM. MILP models have been solved using Gurobi 7 with a single thread. We used irace in version 2.1 for parameter tuning, see López-Ibáñez et al. [2016].

The results of the test instances are grouped by difficulty and number of activities. Unless otherwise indicated, computation times are stated using the median and for all other properties we use the mean. Let *pb* denote the primal bound and *db* the dual bound of the investigated algorithm. The starred versions denote the respective best bounds obtained across all algorithms. Optimality gaps are computed by $100 \cdot \frac{pb-db^*}{db^*}$. Primal bounds are compared using $100 \cdot \frac{pb-pb^*}{pb^*}$ and dual bounds are compared using $100 \cdot \frac{db^*-db}{db^*}$.

We first deal with the parametrization of the primal heuristics used within ITBRA. Then, we compare different combinations of refinement strategies for use within the matheuristic. Finally, we compare ITBRA to a simple metaheuristic and the reference MILP models.

8.2.1 Parametrization of the Primal Heuristics

GRASP from Section 6.2 can also be applied outside the context of the matheuristic, thus, as stand-alone algorithm for SI-PTPSP, when simply applied using an empty initial schedule. We start by explaining how the involved parameters are set since they serve as basis for deriving appropriate values for use within the matheuristic.

The stand-alone GRASP terminates if a time limit of two hours is reached. We chose this criterion primarily to match the time limit of the other approaches, a reasonable degree of convergence is usually reached much earlier. Parameter $k^{\text{grand}}_{\text{ABCH}}$ has been set to 8 for all benchmark instances. We applied irace to determine this value. However, it turned out that the performance of our GRASP is very robust against changes to $k^{\text{grand}}_{\text{ABCH}}$.

For the GRASP embedded in ITBRA we imposed a time limit of 300 seconds and a maximal number of 10,000 iterations without improvement. The latter is set high enough to be non-restrictive in most cases but avoid wasting time if the algorithm already converged sufficiently. The values of the parameters $k^{\text{grand}}_{\text{GCH}}$ and $k^{\text{grand}}_{\text{ABCH}}$ of the embedded GRASP have been determined experimentally starting with the values from the stand-alone variant. For the parameter $k^{\text{grand}}_{\text{GCH}}$ we first assumed a value of $k^{\text{grand}}_{\text{GCH}} = 5 \cdot k^{\text{grand}}_{\text{ABCH}}$ as all activity chains in the test instances consist of five activities. Afterwards, we fine-tuned these parameters by iterative adjustment. The parameter $k^{\text{grand}}_{\text{ABCH}}$ is set to 6 and $k^{\text{grand}}_{\text{GCH}}$ is set to 35. The randomization itself is based on a fixed seed. Tests showed that the chosen termination criteria provide a reasonable balance between result quality and execution speed. Objective values obtained from the embedded GRASP are on average only 0.21% larger tan those obtained from the stand-alone variant. The embedded GRASP provides on average solutions with 16.7% smaller objective value than ABCH.

The local search uses a best improvement strategy. Preliminary experiments confirmed that this strategy works slightly better than a first improvement strategy since the aggregation in terms of activity blocks typically results in only few moves with improvement potential. For the same reason the local optimum is usually reached after a few iterations. Thus, the overhead of the best improvement strategy is not that large. The locally optimal solutions obtained by the best improvement strategy, however, turned out to pay off in terms of a better average quality that is achieved. Tests with irace confirmed this observation, although the differences are quite small. However, for instances with different properties this might not be the case. For a larger number of activity blocks a first improvement strategy might be superior.

8.2.2 Comparison of Bucket Refinement Strategies

Due to the large number of possible combinations of refinement techniques (including further ones not presented in this work) we did not test every variant. Instead, we employed a local search strategy to identify good options. Experiments with the matheuristic terminate if optimality is proven or the time limit of two hours is reached.

We started with variant ASEL,B,UR and then step by step investigated the impact of exchanging each of the three parts, making use of statistical tests. It turned out that the best refinement strategies are VDUE,B,CPR and VDUE,SET+B,CPR. In addition to the these variants we also consider ASEL,B,UR and the best strategy based on AIGS (AIGS,SET,CPR) in the following. The former mainly serves as naïve reference strategy. The latter is used to discuss certain particularities of the bucket refinement process. We shortly summarize the made observations here and refer to Jatschka [2017] for a more detailed discussion.

We compared the four strategies in a pairwise fashion checking the assumption that one strategy yields smaller optimality gaps than the other by a one-tailed Wilcoxon rank-sum test with a significance level of 0.05 per difficulty setting and in total. All considered algorithms perform significantly better than the reference strategy on both

Table 2: Comparison of selected bucket refinement strategies. We consider the average optimality gaps (gap), the number of solved instances (opt) and the median computation times (t). Entries marked with “tl” indicate that the experiment terminated due to the time limit. The summary is obtained by aggregating over the preceding rows using the same function as for the respective column.

	ASEL B UR			AIGS SET CPR			VDUE B CPR			VDUE SET+B CPR		
set	gap[%]	opt	$t[s]$	gap[%]	opt	$t[s]$	gap[%]	opt	$t[s]$	gap[%]	opt	$t[s]$
e_{20}	**0.0**	**15**	**12**	**0.0**	**15**	13	**0.0**	**15**	27	**0.0**	**15**	13
e_{30}	0.6	14	219	0.6	14	**36**	**0.0**	**15**	91	0.6	14	66
e_{40}	5.3	**10**	836	**4.3**	**10**	268	4.9	**10**	200	5.6	**10**	**160**
e_{50}	1.1	14	189	1.1	14	220	**0.0**	**15**	183	**0.0**	**15**	**105**
e_{60}	**1.2**	**14**	82	**1.2**	**14**	**54**	**1.2**	**14**	100	**1.2**	**14**	78
e_{70}	2.1	8	6957	1.2	11	2664	0.4	13	742	**0.3**	**14**	**528**
e_{80}	2.0	7	tl	1.7	9	1687	0.8	10	1197	**0.4**	**13**	**266**
e_{90}	1.7	6	tl	1.6	8	4090	1.6	6	tl	**1.5**	**9**	**1908**
e_{100}	0.9	5	tl	1.3	6	tl	**0.7**	7	tl	1.2	**8**	**4740**
summary	1.7	93	836	1.4	101	268	**1.1**	105	200	1.2	**112**	**160**
h_{20}	**0.0**	**15**	17	**0.0**	**15**	**10**	**0.0**	**15**	22	**0.0**	**15**	20
h_{30}	11.4	9	4341	8.0	10	1726	**6.4**	**12**	1860	**6.4**	**12**	**985**
h_{40}	14.4	4	tl	10.7	6	tl	9.5	6	tl	**8.9**	**7**	tl
h_{50}	18.3	2	tl	18.7	3	tl	**17.0**	**4**	tl	18.2	**4**	tl
h_{60}	18.0	1	tl	17.5	3	tl	17.6	3	tl	**16.4**	**4**	tl
h_{70}	21.9	0	tl	21.0	0	tl	21.2	0	tl	**20.8**	**3**	tl
h_{80}	13.0	1	tl	12.9	1	tl	13.0	1	tl	**12.6**	**2**	tl
h_{90}	11.1	**1**	tl	10.9	**1**	tl	**10.4**	**1**	tl	10.6	**1**	tl
h_{100}	**10.6**	**0**	tl	**10.6**	**0**	tl	**10.6**	**0**	tl	**10.6**	**0**	tl
summary	13.2	33	tl	12.3	39	tl	11.7	42	tl	**11.6**	**48**	tl

instance groups and also in total. The two VDUE variants outperform AIGS on the easy set of instances and in total. However, VDUE,B,CPR is not significantly better than the AIGS variant on the hard set of instances. The VDUE algorithms perform quite similar and none can be shown to work significantly better than the other.

Table 2 provides the results of the selected matheuristic variants. VDUE combined with SET+B and CPR is clearly the dominant strategy when taking computation times into account but is closely followed by VDUE,B,CPR. To discuss the results in depth we present more specific characteristics of the matheuristic variants in Table 3. In particular, we consider the increase in the number of buckets and the average computation time spent per iteration. The former is considered as ratio between the final and the initial number of buckets. The higher this ratio, the more buckets were needed to solve the instance.

Reference strategy ASEL,B,UR generates significantly more buckets than the remaining approaches. This typically keeps the number of iterations low. However, this is paid for excessively in terms of higher computation times per iteration due to the fast increase in model size. In general, the number of buckets grows too fast and unguided to obtain a successful approach.

One could expect AIGS to require the fewest buckets among the introduced strategies

Table 3: Comparison of the characteristics of selected bucket refinement strategies. We consider the ratio between the number of buckets at the start and at the end of the algorithm ($\mathit{ratio}^{\mathrm{B}}$), the average number of iterations (n^{it}), and the average computation time spent per iteration (t^{it}). Column $|B^{\mathrm{init}}|$ provides the average number of buckets contained in the initial bucket partitioning. The summary is obtained by aggregating over the preceding rows using the same function as for the respective column.

		ASEL			AIGS			VDUE			VDUE		
		B			SET			B			SET+B		
		UR			CPR			CPR			CPR		
set	$\lvert B^{\mathrm{init}}\rvert$	ratio$^{\mathrm{B}}$	n^{it}	$t^{\mathrm{it}}[s]$	ratio$^{\mathrm{B}}$	n^{it}	$t^{\mathrm{it}}[s]$	ratio$^{\mathrm{B}}$	n^{it}	$t^{\mathrm{it}}[s]$	ratio$^{\mathrm{B}}$	n^{it}	$t^{\mathrm{it}}[s]$
e_{20}	43	4.73	9	5	2.73	**5**	**2**	1.93	16	**2**	**1.69**	9	**2**
e_{30}	44	7.30	**9**	85	5.77	**9**	62	**2.89**	24	**17**	2.91	16	22
e_{40}	47	7.14	**7**	454	5.75	**7**	330	2.99	19	**158**	**2.91**	13	191
e_{50}	45	8.53	**7**	152	10.13	9	147	**2.83**	15	**25**	2.87	12	33
e_{60}	49	5.30	**4**	218	5.09	5	189	2.50	11	**72**	**2.30**	6	81
e_{70}	49	10.18	**6**	573	8.25	7	395	3.59	18	**89**	**3.52**	12	103
e_{80}	52	7.45	**4**	629	7.01	5	380	3.35	13	131	**3.23**	9	**118**
e_{90}	52	6.94	**3**	809	6.08	4	565	3.54	11	270	**3.53**	8	**235**
e_{100}	58	7.69	**3**	951	6.62	4	708	3.99	13	**266**	**3.75**	9	312
summary	48	7.25	**6**	431	6.38	**6**	309	3.07	16	**114**	**2.97**	10	122
h_{20}	43	4.25	**8**	8	3.73	**8**	6	**1.93**	17	**3**	2.00	13	**3**
h_{30}	44	6.84	**9**	418	6.13	**9**	295	**3.08**	23	121	3.37	19	**109**
h_{40}	43	6.98	**6**	924	6.63	8	621	3.39	17	**276**	**3.29**	12	313
h_{50}	48	5.27	**3**	1812	4.90	5	1160	**2.93**	11	**743**	3.06	9	820
h_{60}	44	5.49	**2**	1926	6.22	5	1166	**3.50**	11	**803**	3.59	9	855
h_{70}	48	4.86	**1**	2629	3.96	3	2611	**3.08**	7	**1280**	3.17	6	1332
h_{80}	49	4.94	**1**	2362	4.97	3	1489	**3.01**	7	**1043**	3.15	5	1112
h_{90}	54	4.97	**1**	2239	4.61	3	1538	**3.04**	6	**1017**	3.22	5	1161
h_{100}	55	4.96	**1**	2617	4.79	3	1648	**2.86**	4	**1287**	3.02	4	1430
summary	48	5.40	**4**	1659	5.10	5	1170	**2.98**	11	**730**	3.10	9	793

due to the potentially small number of refinement candidates. However, using only buckets related to activities scheduled by GCH turned out to be too restrictive. This strategy causes some important splits to be delayed until the bucket partitioning is rather fine-grained.

VDUE is again a strategy that can be expected to generate only few new buckets per iteration. However, compared to AIGS their choice appears to be much more meaningful. Nevertheless, splitting only few buckets leads to a high number of iterations. Fortunately, this is not too problematic due to the small computation times per iteration. Identifying splitting positions with the pure binary strategy leads to only few bucket splits which proves to be beneficial. As SET+B typically selects more candidates, one could expect this strategy to be inferior. However, this is compensated for by incorporating more information obtained from the TBR solution.

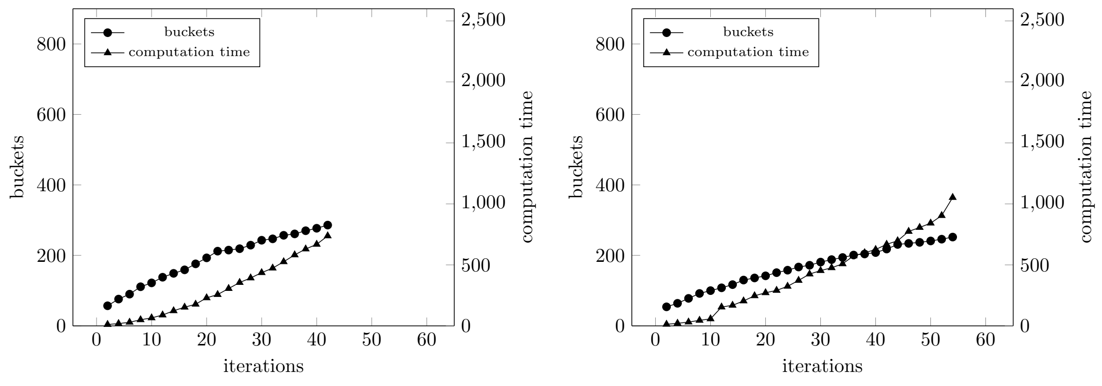
In general, it can be observed that the number of applied splits has a strong influence on performance. However, the quality of the bucket refinement is also very important. For an illustration see Fig. 5. As mentioned before, the large number of buckets generated by ASEL raises the computation time within a few iterations to a problematic level causing an overall bad performance. VDUE,B,CPR features the smallest increase in buckets but requires more iterations to converge. Here it becomes clearly visible that SET+B excels by incorporating more knowledge for making its decision.

Finally, we also want to discuss the properties of the remaining variants in excerpts. Fig. 6 shows a comparison of the average number of iterations and the average final number of buckets for a broad selection of refinement strategies on the set of easy instances with 30 activities. A successful approach is typically characterized by being able to solve an instance by refining only relatively few buckets. Variants that generate many buckets within few iterations usually do not work well. Observe that ASEL and the AIGS variants are all located in the upper half of the figure. The superior strategies are situated near the bottom. It is also clearly visible that SET+B allows to solve an instance in fewer iterations than the pure binary variant. UR and BR are able to solve an instance using fewer buckets and iterations than CPR. This is a peculiarity of the small instances considered here that does not generalize to the larger ones.

8.2.3 Comparing ITBRA to Other Algorithms

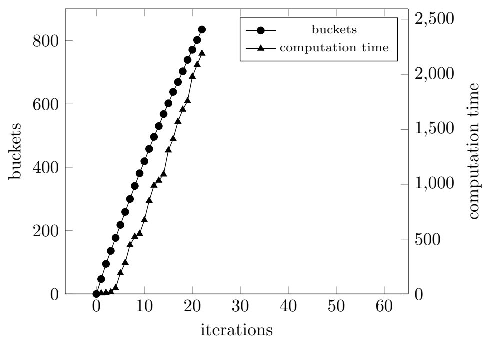
We start by comparing the matheuristic to the stand-alone GRASP, see Table 4. ITBRA is in general able to provide better results. However, when dealing with the most difficult instances, it is sometimes the case that the matheuristic only completes very few iterations and GRASP is able to compute a slightly better solution. As the number of activities increases, ITBRA struggles more and more to improve upon the initially obtained primal bound. This is caused by the originally high computation times per iteration that prevent the algorithm from reaching a sufficient degree of convergence. Remember, however, that ITBRA also puts much effort in determining good lower bounds which GRASP cannot provide at all. In the remainder of this section we compare ITBRA to the compact reference models DEF and TIF.

DEF was not able to find a primal solution for any instance but at least always



(a) VDUE,SET+B,CPR

(b) VDUE,B,CPR



(c) ASEL,B,UR

Figure 5: Comparison of the relation between computation time and increase in the number of buckets for the same e_{40} instance when using different bucket refinement strategies.

Table 4: Comparison of the best found refinement strategies with GRASP. For each algorithm the average gaps to the best primal bound (gap), the standard deviation of the gaps (σ), and the median computation times (t) are presented. Entries marked with “tl” indicate the termination of the experiment due to the time limit. For GRASP, we also provide the number of instances for which a feasible solution could be computed (feas). For the calculation of the gaps we considered only instances for which all algorithms were able to compute a primal bound. The summary is obtained by aggregating over the preceding rows using the same function as for the respective column.

	VDUE B CPR			VDUE SET+B CPR			GRASP		
set	gap[%]	σ	$t[s]$	gap[%]	σ	$t[s]$	gap[%]	σ	feas
e_{20}	**0.0**	**0.0**	27	**0.0**	**0.0**	**13**	38.6	17.3	**12**
e_{30}	**0.0**	**0.0**	91	0.6	2.1	**66**	28.0	16.0	**14**
e_{40}	**3.5**	**7.4**	200	4.2	**7.4**	**160**	12.6	7.7	**15**
e_{50}	**0.0**	**0.0**	183	**0.0**	**0.0**	**105**	7.8	8.0	**15**
e_{60}	**0.8**	**2.9**	100	**0.8**	**2.9**	**78**	3.4	4.6	**15**
e_{70}	0.4	1.6	742	**0.3**	**1.0**	**528**	3.8	3.9	**15**
e_{80}	0.5	1.1	1197	**0.1**	**0.4**	**266**	2.2	3.7	**15**
e_{90}	0.8	**1.0**	tl	**0.7**	**1.0**	**1908**	0.9	**1.0**	**15**
e_{100}	**0.2**	**0.5**	tl	0.7	1.5	**4740**	1.3	2.1	**15**
summary	**0.7**	**1.6**	200	0.8	1.8	**160**	11.0	7.1	**131**
h_{20}	**0.0**	**0.0**	22	**0.0**	**0.0**	**20**	23.3	12.7	**12**
h_{30}	**5.9**	13.1	1860	**5.9**	13.1	**985**	34.9	**11.9**	**15**
h_{40}	6.5	8.2	tl	**5.9**	**8.1**	tl	21.6	11.6	**15**
h_{50}	**6.6**	7.5	tl	7.6	7.9	tl	10.6	**7.1**	**15**
h_{60}	6.9	5.8	tl	**5.6**	**5.5**	tl	8.4	5.6	**15**
h_{70}	2.7	**3.2**	tl	**2.2**	3.3	tl	4.5	5.9	**15**
h_{80}	1.3	2.9	tl	**0.9**	**2.0**	tl	1.1	2.9	**15**
h_{90}	**2.0**	**3.4**	tl	2.2	4.1	tl	2.6	6.0	**15**
h_{100}	1.0	0.9	tl	1.0	0.9	tl	**0.1**	**0.5**	**15**
summary	3.7	**5.0**	tl	**3.5**	**5.0**	tl	11.9	7.1	**132**

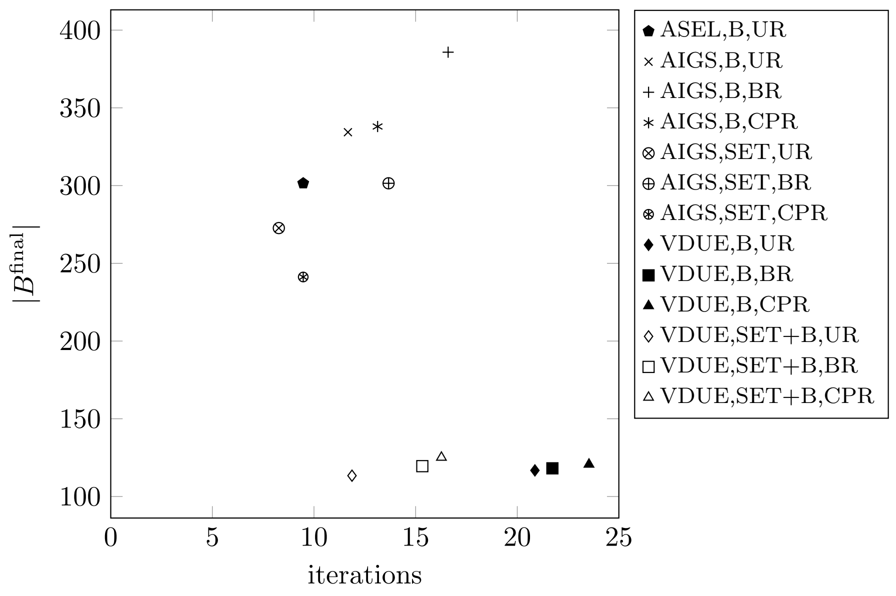


Figure 6: Comparison of the average number of iterations and the average final number of buckets on the set of e_{30} instances.

computed a dual bound. Table 5 provides the comparison with the matheuristic. The bounds obtained from DEF are always worse than those found by ITBRA and turned out to be particularly weak for the group of hard instances, which can be expected due to the looser restrictions featured in this instance group.

As a result of the extremely large time horizons and the memory restriction of 4GB, none of the TIF models even fit into the RAM. Therefore, we consider coarsened TIF models by only taking a subset of the original time horizon into account. Let $\kappa \in \mathbb{N}_{>1}$ be the coarsening measure and TIF^{κ} the associated model. Then, the new time horizon T^{κ} of TIF^{κ} is defined as $T^{\kappa} = \{t \in T \mid t \equiv 0 \pmod{\kappa}\}$. Consequently, we obtain reduced sets of feasible starting times $T_a^{\kappa} = T_a \cap T^{\kappa}$ for the activities $a \in A$. Reducing the number of considered time slots decreases the size of the model, which leads to faster computation times. However, an optimal solution to TIF^{κ} is in general not optimal w.r.t. the original problem due to the disregarded time slots, making it a heuristic approach. A coarsened model might even become infeasible when discarding too many time slots.

Table 6 provides the results of the differently coarsened TIF models. We increase the value of κ stepwise until all instances can either be solved within the time limit or do not permit feasible solutions anymore. For $\kappa < 100$ the models fail to generate a primal bound for almost all instances due to memory or time limitations. Missing table entries (marked with “-”) indicate that the coarsened model is not able to find a primal bound for any instance of the corresponding set. For smaller instances the TIF^{κ} models are able to produce reasonable primal solutions. However, the quality of the solutions deteriorates drastically as more time slots are disregarded. No TIF^{κ} variant

Table 5: Comparison between ITBRA and DEF. For each algorithm we provide the average gaps to the best dual bound (gap), the standard deviation of the gaps (σ) and the median computation times (t). Entries marked with "tl" indicate the termination of the experiment due to the time limit. The summary is obtained by aggregating over the preceding rows using the same function as for the respective column.

	VDUE			VDUE					
	B			SET+B					
	CPR			CPR			DEF		
set	gap[%]	σ	$t[s]$	gap[%]	σ	$t[s]$	gap[%]	σ	$t[s]$
e_{20}	**0.0**	**0.0**	27	**0.0**	**0.0**	**13**	15.3	10.6	tl
e_{30}	**0.0**	**0.0**	91	**0.0**	0.1	**66**	7.6	6.5	tl
e_{40}	**0.2**	**0.4**	200	**0.2**	0.5	**160**	4.5	7.8	tl
e_{50}	**0.0**	**0.0**	183	**0.0**	**0.0**	**105**	3.0	6.4	tl
e_{60}	**0.0**	**0.0**	100	**0.0**	**0.0**	**78**	1.8	3.2	tl
e_{70}	**0.0**	**0.0**	742	**0.0**	**0.0**	**528**	1.6	2.4	tl
e_{80}	**0.0**	**0.0**	1197	**0.0**	0.1	**266**	1.0	1.7	tl
e_{90}	**0.0**	**0.1**	tl	0.1	0.2	**1908**	1.5	2.8	tl
e_{100}	**0.0**	**0.1**	tl	0.1	**0.1**	**4740**	2.0	1.8	tl
summary	**0.0**	**0.1**	200	**0.0**	**0.1**	**160**	4.3	4.8	tl
h_{20}	**0.0**	**0.0**	22	**0.0**	**0.0**	**20**	19.6	11.4	tl
h_{30}	**0.3**	**1.1**	1860	0.4	**1.1**	**985**	31.3	12.2	tl
h_{40}	0.2	0.4	tl	**0.1**	**0.2**	tl	18.6	13.3	tl
h_{50}	2.0	3.3	tl	**1.6**	**1.5**	tl	15.4	9.0	tl
h_{60}	**0.9**	**1.7**	tl	1.3	2.1	tl	4.3	4.8	tl
h_{70}	**2.8**	4.0	tl	2.9	**3.9**	tl	6.7	7.4	tl
h_{80}	1.5	**4.8**	tl	**1.4**	4.9	tl	2.3	4.9	tl
h_{90}	**0.3**	0.7	tl	**0.3**	**0.6**	tl	3.8	5.9	tl
h_{100}	**0.3**	**0.4**	tl	**0.3**	**0.4**	tl	1.0	1.3	tl
summary	**0.9**	1.8	tl	**0.9**	**1.6**	tl	11.4	7.8	tl

is able to find a primal solution for all instances. When using a small value for κ, many instances cannot be solved due to the time limit. For larger κ-values we can solve more instances but at the cost of much larger gaps. Moreover, as we reduce the precision even further, the models start to become infeasible. The number of infeasible instances strongly increases for $\kappa \geq 10000$ and the few instances that still permit feasible solutions feature gaps of over 120%. Therefore, further increasing the value of κ does not seem meaningful. It appears that there does not exist an appropriate value for κ allowing a reasonable balance between computation time and result quality. Due to the many missing entries we decided to use median instead of average gaps in the summary table.

According to our experiments the best variants are those with $\kappa = 1000$ and $\kappa = 2000$, respectively. The former provides better solutions but the latter is able to find more feasible solutions. For some instances the coarsened TIF variants even find better primal solutions than the matheuristic. Especially for instance sets h_{40}, h_{50}, and h_{60} we obtain a high number of good solutions s.t. also the median gaps are smaller here. Overall, however, ITBRA still provides the better results. Moreover, recall that the TIF^{κ} models can only provide heuristic solutions and no dual bounds.

Last but not least, we also investigated the use of disaggregated precedence constraints (see Artigues [2017]) but this did not lead to significant improvements.

9 Conclusions

In this work we considered a matheuristic, referred to as iterative time-bucket refinement algorithm (ITBRA), intended for solving a resource-constrained project scheduling problem (RCPSP) that requires scheduling in high resolution. We proposed a relaxation for the original problem based on aggregating consecutive integral time points into so-called time-buckets. Exploiting this relaxation we constructed a matheuristic that solves this relaxation based on iteratively refined bucket partitionings. Moreover, we heuristically derive primal bounds incorporating information from the relaxed solution. The matheuristic then attempts to close the gap between dual bounds obtained from the relaxation and primal bounds determined by (meta-)heuristics. The crucial part of this approach is how to determine the subsequent (more refined) bucket partitioning for the next iteration. We considered a variety of strategies and investigated them on a novel benchmark set motivated by an application arising in particle therapy for cancer treatment.

Our experiments indicate that it is most critical to limit the increase in the number of buckets. However, the quality of the applied bucket splits has a substantial impact on the convergence speed. Strategy VDUE,SET+B,CPR turned out to work best in this respect.

The matheuristic works better than a simple greedy randomized adaptive search procedure (GRASP) on all instance sets except for the most difficult one. There it fails to complete a sufficient number of iterations to make reasonable improvements to the primal bound.

ITBRA clearly outperforms the compact mixed integer linear programming (MILP)

Table 6: Comparison of differently coarsened TIF models with ITBRA. We provide the median gaps to the best primal bound of the original problem (gap) and the median computation times (t). Missing entries ("-") indicate that the coarsened model is not able to find a primal bound for any instance of the corresponding set. Moreover, for each instance set we indicate the number of optimally (opt^c) and feasibly (feas) solved instances w.r.t. the coarsened model. Column *infeas* denotes the number of instances with proven infeasible model. Finally, we indicate the number of instances that terminated due to the time limit (tl) or the memory limit (ml), respectively. The summary is obtained by aggregating over the preceding rows using the same function as for the respective column.

	TIF^{100}		TIF^{200}		TIF^{1000}		TIF^{2000}		TIF^{10000}		VDUE B CPR		VDUE SET+B CPR	
set	gap^{med} [%]	$t[s]$	gap^{med} [%]	$t[s]$	gap^{med} [%]	$t[s]$	gap^{med} [%]	$t[s]$	gap^{med} [%]	$t[s]$	gap^{med} [%]	$t[s]$	gap^{med} [%]	$t[s]$
e_{20}	0.3	15	0.6	4	21.7	**<1**	24.0	**<1**	-	-	**0.0**	27	**0.0**	13
e_{30}	0.1	310	0.4	58	4.0	4	18.3	**2**	-	-	**0.0**	91	**0.0**	66
e_{40}	0.3	2940	0.5	698	3.8	40	12.3	**6**	-	-	**0.0**	200	**0.0**	160
e_{50}	0.2	409	0.4	113	1.9	17	6.4	5	-	**<1**	**0.0**	183	**0.0**	105
e_{60}	0.3	5569	0.4	839	2.4	52	4.7	21	-	**<1**	**0.0**	100	**0.0**	78
e_{70}	14.3	tl	0.3	2713	1.9	165	4.7	77	-	**<1**	**0.0**	742	**0.0**	528
e_{80}	-	tl	0.3	5630	1.7	292	3.4	108	-	**<1**	**0.0**	1197	**0.0**	266
e_{90}	-	tl	-	tl	0.9	555	3.0	327	-	**1**	**0.0**	tl	**0.0**	1908
e_{100}	-	tl	-	tl	1.8	652	4.0	263	-	**6**	**0.0**	tl	**0.0**	4740
summary	0.3	5569	0.4	839	1.9	52	4.7	21	-	**<1**	**0.0**	200	**0.0**	160
h_{20}	0.4	39	0.5	8	6.4	1	19.8	**<1**	-	**<1**	**0.0**	22	**0.0**	20
h_{30}	11.8	6106	1.0	1129	11.1	42	24.6	**13**	-	-	**0.0**	1860	**0.0**	985
h_{40}	35.4	tl	**0.6**	tl	5.5	227	13.7	65	-	**<1**	3.9	tl	3.2	tl
h_{50}	-	tl	-	tl	**0.0**	2815	9.5	381	-	**<1**	3.5	tl	3.9	tl
h_{60}	-	tl	8.9	tl	**2.3**	1532	7.0	940	-	**<1**	9.2	tl	6.5	tl
h_{70}	-	tl	-	tl	14.3	tl	10.4	3052	82.5	**<1**	1.6	tl	**0.0**	tl
h_{80}	-	tl	-	tl	5.0	tl	12.1	tl	77.0	**3**	**0.0**	tl	**0.0**	tl
h_{90}	-	tl	-	tl	9.0	tl	14.2	tl	93.8	**8**	**0.3**	tl	**0.3**	tl
h_{100}	-	tl	-	tl	39.6	tl	22.7	tl	-	**16**	**1.2**	tl	**1.2**	tl
summary	-	tl	-	tl	6.4	2815	13.7	940	-	**<1**	1.2	tl	**0.3**	tl

	TIF^{100}					*TIF*^{200}				*TIF*^{1000}				*TIF*^{2000}				*TIF*^{10000}			
set	opt^c	feas	infeas	tl	ml	opt^c	feas	infeas	tl	opt^c	feas	infeas	tl	opt^c	feas	infeas	tl	opt^c	feas	infeas	tl
e_{20}	**15**	**15**	**0**	**0**	**0**	**15**	**15**	**0**	**0**	**15**	**15**	**0**	**0**	13	13	2	**0**	0	0	15	**0**
e_{30}	**15**	**15**	**0**	**0**	**0**	**15**	**15**	**0**	**0**	**15**	**15**	**0**	**0**	**15**	**15**	**0**	**0**	0	0	15	**0**
e_{40}	9	12	**0**	6	**0**	12	14	**0**	3	**15**	**15**	**0**	**0**	**15**	**15**	**0**	**0**	0	0	15	**0**
e_{50}	13	14	**0**	2	**0**	14	14	**0**	1	**15**	**15**	**0**	**0**	**15**	**15**	**0**	**0**	2	2	13	**0**
e_{60}	9	9	**0**	6	**0**	14	**15**	**0**	1	**15**	**15**	**0**	**0**	**15**	**15**	**0**	**0**	6	6	9	**0**
e_{70}	6	9	**0**	9	**0**	12	14	**0**	3	**15**	**15**	**0**	**0**	**15**	**15**	**0**	**0**	2	2	13	**0**
e_{80}	3	3	**0**	12	**0**	9	11	**0**	6	**15**	**15**	**0**	**0**	**15**	**15**	**0**	**0**	7	7	8	**0**
e_{90}	0	0	**0**	13	**2**	6	7	**0**	9	**14**	14	**0**	1	**14**	**15**	**0**	1	4	4	11	**0**
e_{100}	1	1	**0**	8	**6**	4	5	**0**	11	13	**15**	**0**	2	**15**	**15**	**0**	**0**	2	2	13	**0**
summary	71	78	**0**	56	**8**	101	110	**0**	34	**132**	**134**	**0**	3	**132**	133	2	1	23	23	112	**0**
h_{20}	**15**	**15**	**0**	**0**	**0**	**15**	**15**	**0**	**0**	**15**	**15**	**0**	**0**	**15**	**15**	**0**	**0**	2	2	13	**0**
h_{30}	8	12	**0**	7	**0**	14	**15**	**0**	1	**15**	**15**	**0**	**0**	14	14	1	**0**	0	0	15	**0**
h_{40}	4	9	**0**	11	**0**	8	12	**0**	7	**15**	**15**	**0**	**0**	**15**	**15**	**0**	**0**	5	5	10	**0**
h_{50}	1	4	**0**	14	**0**	4	7	**0**	11	12	**15**	**0**	3	**15**	**15**	**0**	**0**	6	6	9	**0**
h_{60}	0	0	**0**	15	**0**	2	9	**0**	13	9	**15**	**0**	6	**14**	**15**	**0**	1	6	6	9	**0**
h_{70}	0	0	**0**	15	**0**	0	1	**0**	15	4	10	**0**	11	**10**	**15**	**0**	5	8	8	7	**0**
h_{80}	0	0	**0**	11	**4**	1	4	**0**	14	5	11	**0**	10	4	**15**	**0**	11	**9**	9	6	**0**
h_{90}	0	0	**0**	12	**3**	1	1	**0**	14	4	**12**	**0**	11	6	**12**	**0**	9	**8**	8	7	**0**
h_{100}	0	0	**0**	1	**14**	0	0	**0**	15	1	8	**0**	14	0	**11**	**0**	15	**5**	5	10	**0**
summary	28	40	**0**	86	**21**	45	64	**0**	90	80	116	**0**	55	**93**	**127**	1	41	49	49	86	**0**

formulations. The considered discrete-event formulation (DEF) is only capable of computing dual bounds for all of our benchmark instances and the considered time-indexed formulation (TIF) cannot even be solved due to its model size. Variants of TIF based on a coarsened time horizon are manageable but become infeasible once too many time points are disregarded. For some instances good primal solutions could be obtained but there exists no coarsening factor that works well in general by providing a good balance between model size and result quality.

We primarily focused on MILP-based algorithms here. Another well-known exact technique often used to deal with scheduling problems is constraint programming (CP). In a more comprehensive study it appears to be interesting to compare our matheuristic also to a suitable CP approach. Moreover, it might also be relevant to consider CP techniques within ITBRA to improve its performance. In general the (meta-)heuristics currently used within the matheuristic are rather simple. In particular, they suffer from the effects of fixing the time lags which prevents them from considering a large variety of possible solutions. This is a crucial part of the matheuristic for which more elaborate techniques should be identified and tested.

In the computational study we investigated the power of our algorithm on a rather specific set of benchmark instances. The fundamental approach, however, is in principle much more generally applicable to problems that require scheduling in high resolution. To verify this a more diversified set of benchmark instances, originating from different application domains, has to be considered. Of course this requires adjusted MILP formulations and adapted as well as novel bucket refinement strategies.

Acknowledgments

We thank EBG MedAustron GmbH, Wiener Neustadt, Austria, for the collaboration on particle therapy patient scheduling and partially funding this work.

References

C. Artigues. On the strength of time-indexed formulations for the resource-constrained project scheduling problem. *Operations Research Letters*, 45(2):154 – 159, 2017.

C. Artigues and E. Hebrard. Mip relaxation and large neighborhood search for a multi-mode resource-constrained multi-project scheduling problem. In G. Kendall, B. McCollum, and G. Venden Berghe, editors, *In proceedings of the 6th Multidisciplinary International Conference on Scheduling : Theory and Applications (MISTA 2013), 27–30 Aug 2013, Ghent, Belgium*, pages 815–819, 2013.

C. Artigues, S. Demassey, and E. Neron. *Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications.* Wiley-ISTE, 2008.

C. Artigues, P. Brucker, S. Knust, O. Koné, and P. Lopez. A note on "event-based

MILP models for resource-constrained project scheduling problems". *Computers and Operations Research*, 40(4):1060–1063, 2013.

P. Baptiste and R. Sadykov. On scheduling a single machine to minimize a piecewise linear objective function: A compact MIP formulation. *Naval Research Logistics*, 56 (6):487–502, 2009.

G. Baydoun, A. Haït, R. Pellerin, B. Clément, and G. Bouvignies. A rough-cut capacity planning model with overlapping. *OR Spectrum*, 38(2):335–364, 2016.

T. Berthold, S. Heinz, M. E. Lübbecke, R. H. Möhring, and J. Schulz. A constraint integer programming approach for resource-constrained project scheduling. In *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*, pages 313–317. Springer, 2010.

L. Bianco and M. Caramia. A new lower bound for the resource-constrained project scheduling problem with generalized precedence relations. *Computers & Operations Research*, 38(1):14–20, 2011a.

L. Bianco and M. Caramia. Minimizing the completion time of a project under resource constraints and feeding precedence relations: a lagrangian relaxation based lower bound. *4OR*, 9(4):371–389, 2011b.

L. Bianco and M. Caramia. An exact algorithm to minimize the makespan in project scheduling with scarce resources and generalized precedence relations. *European Journal of Operational Research*, 219(1):73–85, 2012.

L. P. Bigras, M. Gamache, and G. Savard. Time-indexed formulations and the total weighted tardiness problem. *INFORMS Journal on Computing*, 20(1):133–142, 2008.

N. Boland, R. Clement, and H. Waterer. A Bucket Indexed Formulation for Nonpreemptive Single Machine Scheduling Problems. *INFORMS Journal on Computing*, 28 (1):14–30, 2016.

N. Boland, M. Hewitt, L. Marshall, and M. Savelsbergh. The continuous-time service network design problem. *Operations Research*, 65(5):1303–1321, 2017.

C. Bron and J. Kerbosch. Algorithm 457: Finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, 1973.

P. Brucker, A. Drexl, R. Möhring, K. Neumann, and E. Pesch. Resource-constrained project scheduling: Notation, classification, models, and methods. *European journal of operational research*, 112(1):3–41, 1999.

J. Carlier, A. Moukrim, and A. Sahli. Lower bounds for the Event Scheduling Problem with Consumption and Production of Resources. *Discrete Applied Mathematics*, to appear. available at https://doi.org/10.1016/j.dam.2016.05.021.

F. Cazals and C. Karande. A note on the problem of reporting maximal cliques. *Theoretical Computer Science*, 407(1–3):564 – 568, 2008. ISSN 0304-3975.

F. Clautiaux, S. Hanafi, R. Macedo, M. Voge, and C. Alves. Iterative aggregation and disaggregation algorithm for pseudo-polynomial network flow models with side constraints. *European Journal of Operational Research*, 258(2):467–477, 2017.

S. Dash, O. Günlük, A. Lodi, and A. Tramontani. A time bucket formulation for the traveling salesman problem with time windows. *INFORMS Journal on Computing*, 24(1):132–147, 2012.

S. Demassey, C. Artigues, and P. Michelon. Constraint-propagation-based cutting planes: An application to the resource-constrained project scheduling problem. *INFORMS Journal on Computing*, 17(1):52–65, 2005.

E. L. Demeulemeester and W. S. Herroelen. A Branch-and-Bound Procedure for the Generalized Resource-Constrained Project Scheduling Problem. *Operations Research*, 45(2):201–212, 1997.

N. Dupin and E. G. Talbi. Dual Heuristics and New Lower Bounds for the Challenge EURO/ROADEF 2010. In *Matheuristics 2016 - Proceedings of the Sixth International Workshop on Model-based Metaheuristics, 4–7 Sep 2016, Brussels, Belgium*, pages 60–71, 2016.

M. L. Fisher. Optimal Solution of Scheduling Problems Using Lagrange Multipliers: Part I. *Operations Research*, 21(5):1114–1127, 1973.

R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete mathematics*, 5:287–326, 1979.

F. Guerriero and L. Talarico. A solution approach to find the critical path in a time-constrained activity network. *Computers & Operations Research*, 37(9):1557 – 1569, 2010.

J. R. Hardin, G. L. Nemhauser, and M. W. P. Savelsbergh. Strong valid inequalities for the resource-constrained scheduling problem with uniform resource requirements. *Discrete Optimization*, 5(1):19 – 35, 2008. ISSN 1572-5286.

S. Hartmann and D. Briskorn. A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*, 207(1):1–14, 2010.

J. N. Hooker. Planning and Scheduling by Logic-Based Benders Decomposition. *Operations Research*, 55(3):588–602, 2007.

T. Jatschka. An iterative refinement algorithm for high resolution scheduling problems. Master's thesis, TU Wien, 2017. in preparation.

O. Koné, C. Artigues, P. Lopez, and M. Mongeau. Event-based MILP models for resource-constrained project scheduling problems. *Computers & Operations Research*, 38(1):3–13, 2011.

E. L. Lawler and J. K. Lenstra. Machine Scheduling with Precedence Constraints. In I. Rival, editor, *Ordered Sets*, pages 655–675. Springer Netherlands, 1982.

Y. Li, O. Ergun, and G. L. Nemhauser. A dual heuristic for mixed integer programming. *Operations Research Letters*, 43(4):411–417, 2015.

M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

R. Macedo, C. Alves, J. De Carvalho, F. Clautiaux, and S. Hanafi. Solving the vehicle routing problem with time windows and multiple routes exactly using a pseudo-polynomial model. *European Journal of Operational Research*, 214(3):536–545, 2011.

A. K. Mackworth. Consistency in networks of relations. *Artificial intelligence*, 8(1): 99–118, 1977.

V. Maniezzo and A. Mingozzi. A heuristic procedure for the multi-mode project scheduling problem based on Benders' decomposition. In J. Weglarz, editor, *Project Scheduling: Recent Models, Algorithms and Applications*, pages 179–196. Springer US, Boston, MA, 1999.

V. Maniezzo, T. Stützle, and S. Voß, editors. *Matheuristics: Hybridizing Metaheuristics and Mathematical Programming.* Springer US, 2010.

R. H. Möhring, A. S. Schulz, F. Stork, and M. Uetz. Solving project scheduling problems by minimum cut computations. *Management Science*, 49(3):330–350, 2003.

K. Neumann, C. Schwindt, and J. Zimmermann. *Project Scheduling with Time Windows and Scarce Resources.* Springer Berlin Heidelberg, 2003.

M. Palpant, C. Artigues, and P. Michelon. LSSPER: Solving the resource-constrained project scheduling problem with large neighbourhood search. *Annals of Operations Research*, 131(1-4):237–257, 2004.

M. G. C. Resende and C. C. Ribeiro. Greedy randomized adaptive search procedures: Advances, hybridizations, and applications. In M. Gendreau and J. Potvin, editors, *Handbook of Metaheuristics*, pages 283–319. Springer US, Boston, MA, 2010.

T. A. M. Toffolo, H. G. Santos, M. A. M. Carvalho, and J. A. Soares. An integer programming approach to the multimode resource-constrained multiproject scheduling problem. *Journal of Scheduling*, 19(3):295–307, 2016.

X. Wang and A. C. Regan. Local truckload pickup and delivery with hard time window constraints. *Transportation Research Part B: Methodological*, 36(2):97–112, 2002.

X. Wang and A. C. Regan. On the Convergence of a New Time Window Discretization Method for the Traveling Salesman Problem with Time Window Constraints. *Computers & Industrial Engineering*, 56(1):161–164, 2009.

J. Westerlund, M. Hästbacka, S. Forssell, and T. Westerlund. Mixed-Time Mixed-Integer Linear Programming Scheduling Model. *Industrial & Engineering Chemistry Research*, 46(9):2781–2796, 2007.