FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

An Iterative Time-Bucket Refinement Algorithm for High Resolution Scheduling Problems

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Logic and Computation

eingereicht von

Thomas Jatschka, BSc
Matrikelnummer 0928678

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Günther Raidl
Mitwirkung: Univ.-Ass. Dipl.-Ing. Martin Riedler, BSc
Projektass. Dipl.-Ing. Johannes Maschler, BSc

Wien, 11. Oktober 2017

Thomas Jatschka

Günther Raidl

FAKULTÄT
FÜR !NFORMATIK

Faculty of Informatics

An Iterative Time-Bucket Refinement Algorithm for High Resolution Scheduling Problems

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Logic and Computation

by

Thomas Jatschka, BSc
Registration Number 0928678

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Günther Raidl
Assistance: Univ.-Ass. Dipl.-Ing. Martin Riedler, BSc
Projektass. Dipl.-Ing. Johannes Maschler, BSc

Vienna, 11th October, 2017

	Thomas Jatschka	Günther Raidl

Erklärung zur Verfassung der Arbeit

Thomas Jatschka, BSc
Hardtmuthgasse 58/1/6, Wien 1100

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 11. Oktober 2017

Thomas Jatschka

Danksagung

Ich möchte mich herzlich bei meinen Betreuern Günther Raidl, Martin Riedler und Johannes Maschler bedanken. Ich bin ihnen sehr dankbar für ihre großartige Unterstützung und Geduld. Dank ihnen konnte ich viel Neues lernen und die Qualität meiner Arbeit verbessern.

Ich möchte mich auch bei der EBG MedAustron GmbH[1], Marie Curie-Straße 5, 2700 Wiener Neustadt, Österreich, für die Zusammenarbeit und die finanzielle Unterstützung dieser Arbeit bedanken.

Zu guter Letzt, möchte ich mich auch bei meinen Eltern und bei meinem Bruder Johannes dafür bedanken, dass sie mich mein ganzes Leben lang unterstützt haben.

Teile dieser Arbeit wurden veröffentlicht in M. Riedler, T. Jatschka, J. Maschler, and G. R. Raidl. An iterative time-bucket refinement algorithm for a high-resolution resource-constrained project scheduling problem. *International Transactions in Operational Research*, to appear. available at `http://dx.doi.org/10.1111/itor.12445`.

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

Acknowledgements

I would like express my sincere gratitude to Günther Raidl, Martin Riedler, and Johannes Maschler for supervising this thesis. I am very grateful for their great support and patience. Thanks to them I was able to learn many new things and improve the quality of this thesis.

I also want to thank EBG MedAustron GmbH[2], Marie Curie-Straße 5, 2700 Wiener Neustadt, Österreich, for their cooperation and for partially funding this thesis.

Last but not least, I want to thank my parents and my brother Johannes for supporting me throughout my whole life.

Parts of this thesis have been published in M. Riedler, T. Jatschka, J. Maschler, and G. R. Raidl. An iterative time-bucket refinement algorithm for a high-resolution resource-constrained project scheduling problem. *International Transactions in Operational Research*, to appear. available at `http://dx.doi.org/10.1111/itor.12445`.

[2]`https://www.medaustron.at`

Kurzfassung

In dieser Arbeit werden Algorithmen zum Lösen von *Scheduling* Problemen, die einem langen Zeithorizont unterliegen, entwickelt. Diese Algorithmen werden auf ein Problem, das durch ein Patientenplanungsszenario des Krebsbehandlungszentrums MedAustron in Wiener Neustadt, Österreich, motiviert ist, angewandt. Ziel ist es, einen Plan für die individuellen Behandlungstermine der Patienten zu erstellen, sodass zeitliche Abhängigkeiten zwischen den Behandlungen eingehalten werden. Jede Behandlungsphase benötigt verschiedene Ressourcen. Eine dieser Ressourcen ist der Teilchenstrahl, dessen Nutzung insbesondere optimiert werden muss, da er für jede Behandlung benötigt wird und abwechselnd in mehreren Behandlungsräumen eingesetzt wird. Es soll ein Plan erstellt werden, der so dicht wie möglich ist, sodass möglichst viele Patienten behandelt werden können. Außerdem führt ein kompakter Plan zu einer Reduzierung der Standzeit des Teilchenstrahls.

Es werden sowohl exakte als auch heuristische Verfahren entwickelt, um das Problem zu lösen. Als heuristisches Lösungsverfahren wird eine *Greedy Randomized Adaptive Search Procedure* (GRASP) verwendet. Die exakten Algorithmen basieren auf gemischt-ganzzahliger linearer Optimierung (engl. *mixed integer linear programming* (MILP)). Es werden verschiedene MILP-Modelle entwickelt und sowohl in Bezug auf die Modellstärke als auch mithilfe empirischer Experimente miteinander verglichen.

Der Hauptalgorithmus der Arbeit ist eine Matheuristik, die MILP mit heuristischen Ansätzen kombiniert. Die Grundidee besteht darin, das Problem zu lösen, ohne explizit den gesamten Zeithorizont zu berücksichtigen. Stattdessen basiert der Algorithmus auf einem *relaxierten* Modell, in dem der Zeithorizont in sogenannte *time-buckets* partitioniert wird. Dieses reduzierte Modell ist üblicherweise viel kleiner als das ursprüngliche und kann daher relativ schnell gelöst werden. Eine Lösung des relaxierten Problems repräsentiert eine duale Schranke für den tatsächlichen Lösungswert. Bei der Lösung handelt es sich aber üblicherweise nicht um einen gültigen Plan. Daher wird eine Heuristik verwendet, deren Ziel es ist, eine gültige Lösung (primale Schranke) aus der Lösung des *relaxierten* Modells abzuleiten. Darüber hinaus zerteilt der Algorithmus mehrere *time-buckets*, um nach erneutem Lösen des Modells eine bessere Schranke zu erhalten. Die Unterteilung basiert auf Informationen, die aus der Lösung des *relaxierten* Modells gewonnen werden. Durch das iterative Ausführen dieser Prozedur ergibt sich eine Matheuristik, welche schlussendlich zu einer beweisbar optimalen Lösung konvergiert.

Anhand zweier Gruppen neuer Testinstanzen werden verschiedene Strategien zur Unterteilung von *time-buckets* untersucht und ein Vergleich mit anderen exakten und heuristischen Lösungsverfahren durchgeführt.

Abstract

In this thesis algorithms are developed for solving scheduling problems subject to a large time horizon. We apply these algorithms on a problem motivated by a real world patient scheduling scenario at the cancer treatment center MedAustron located in Wiener Neustadt, Austria. The tasks involved in providing a given set of patients with their individual particle treatments shall be scheduled in such a way that given minimum and maximum waiting times 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 goal is to find a schedule which is as dense as possible to allow treating as many patients as possible. Moreover, a dense schedule reduces the idle time of the particle beam within the day.

We develop different exact as well as heuristic algorithms for tackling the problem. A greedy randomized adaptive search procedure (GRASP) is used to heuristically solve the problem. The exact algorithms are based on mixed integer linear programming (MILP). We provide different MILP models and compare the strength of models that are of particular interest.

The main algorithm of this thesis is a matheuristic which combines exact mathematical programming methods as well as heuristic approaches. The basic idea of our matheuristic is to solve the problem without explicitly considering the complete time horizon. Instead, the algorithm considers a relaxed model which is based on partitioning the time horizon into so called time-buckets. This relaxation is typically much smaller than the original model and can be solved relatively quickly. An obtained solution provides a dual bound for the problem's solution value but in general does not represent a feasible schedule. Using the solution to the relaxation, the algorithm tries to heuristically derive a primal bound, i.e., a feasible schedule. Moreover, the algorithm also subdivides some time-buckets based on information gained from the solution to the relaxation and resolves the resulting refined model to obtain an improved bound on the problem. Doing this refinement iteratively yields a matheuristic that in principle converges to a provably optimal solution.

A novel set of test instances is used to evaluate the performance of different refinement strategies of the matheuristic and to compare the matheuristic to other exact and heuristic methods.

Contents

CHAPTER 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 planning 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 troubles as a result of their weak LP bounds.

In the following we focus on problems with these characteristics 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[1] located in Wiener Neustadt, Austria. 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. For a formal definition of the problem see Chapter 4.

The main goal therefore is to exploit in particular the availability of the particle beam as best as possible by suitably scheduling all activities in high resolution. Ideally, the

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

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.

1.1 Structure of the Work

The thesis is organized as follows. In Chapter 2 we review the related literature. Afterwards, in Chapter 3 we describe the methodological concepts used in the thesis. Chapter 4 formally defines the investigated problem and provides different MILP-formulations for solving it. The main part of the thesis is Chapter 5 in which we present our matheuristic. Implementation details are provided in Chapter 6. Afterwards, in Chapter 7, we discuss computational experiments conducted on two sets of benchmark instances. We conclude the thesis with Chapter 8 by giving an outlook on promising future research directions.

CHAPTER 2

State Of The Art

In this chapter we discuss the related work relevant for this thesis. 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 the scheduling domain. Finally, we review previous work dealing with scheduling problems subject to a large time horizon.

2.1 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. For an overview see Kolisch [1995], Brucker et al. [1999], Neumann et al. [2003], and Artigues et al. [2008].

Our problem is a combination of multiple extensions of the RCPSP. One of these extensions is the RCPSP with generalized precedence constraints, extending the RCPSP by minimal and maximal time lags between the end of one activity and the start of another activity, see Bianco and Caramia [2012], Cesta et al. [2002], and De Reyck and Herroelen [1998]. Minimal time lags impose a minimal waiting time between the end and the start of activities. Analogously, maximal time lags impose a maximal waiting time between the end and the start of activities.

Activities can also be subject to release times and deadlines (Bomsdorf and Derigs [2008], Klein [2000], Demeulemeester and Herroelen [1997]), meaning that an activity has to be completely processed within the time window specified by these respective bounds. An RCPSP with release times and deadlines for the activities is referred to as generalized RCPSP (see Klein [2000], Demeulemeester and Herroelen [1997]).

For our problem resources are not always available which is usually referred to as partially renewable resources in project scheduling (see Böttcher et al. [1999]). Note that using release times and deadlines one can model unavailability periods of resources by introducing additional activities (see Bomsdorf and Derigs [2008]).

There exists a wide range of exact and heuristic approaches for the RCPSP and its extensions, for an overview see Brucker et al. [1999], Neumann et al. [2003], and Artigues et al. [2008]. Examples of heuristic approaches can be found in Bomsdorf and Derigs [2008] and Kolisch and Hartmann [2006]. 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 to represent 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 to model 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.

Further MILP techniques for approaching the considered scheduling problems make use of exponentially sized models and apply advanced techniques such as column generation, Lagrangian decomposition, or Benders decomposition, see, e.g., Hooker [2007]. While they are frequently very successful, they are also substantially more complex to develop, implement, and fine-tune.

2.2 Dual Bounds for Scheduling Problems

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

Other techniques for deriving dual bounds based on altering the MILP's constraints are: the constraint relaxation, the Lagrangian relaxation and the surrograte relaxation (see Li et al. [2015]).

The constraint relaxation derives a dual bound of a MILP model by simply dropping some of the model's constraints.

The Lagrangian relaxation dualizes constraints by adding them as a penalty term to the model's objective function. Such a relaxation is presented by Fisher [1973] for a network scheduling problem under resource constraints. Lagrangian relaxation is used in order to dualize the resource constraints.

The third technique is the surrogate relaxation which derives a new constraint by aggregating a set of constraints and replacing the original ones (see Glover [1965]).

A less common method for generating dual bounds is the dual heuristic algorithm by Li et al. [2015]. For some nodes of the B&B tree, the heuristic attempts to improve the current dual bound by computing an additional relaxation, e.g., a constraint or a surrogate relaxation. 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. For an example see Dupin and Talbi [2016]. The contribution deals with fulfilling energy demands over a given time horizon. The energy is provided by power plants which have to be refuelled and maintained regularly. Moreover, during refuelling and maintenance some power plants have to go offline. The objective function is to minimize the expected production costs over a given set of scenarios. The time horizon is split into intervals of the same length, so called time steps. While production periods are planned for each time step, offline periods are scheduled in weeks. Dupin and Talbi [2016] provide different MILPs for computing lower bounds for the production costs. In one such MILP production time steps are aggregated to weekly production periods.

Another problem specific relaxation method is presented by Carlier and Néron [2003] for the RCPSP. The relaxation is formulated as an MILP which is based on a partitioning of the scheduling horizon. However, as the bounds generated by this formulation may be too weak, Carlier and Néron [2003] encode different estimations of the makespan (linear lower bounds (LLB)) into the model as constraints. Each LLB underestimates the makespan and is based on different properties of the problem, e.g., resource capacities or critical paths. The quality of the relaxation is controlled by the number of LLBs added to the model.

Further techniques for generating dual bounds for the RCPSP can be found in Bianco and Caramia [2011].

2.3 Matheuristics for Scheduling Problems

So far, Matheuristics have only been rarely considered to tackle the RCPSP. For an example see Palpant et al. [2004], who developed a large scale neighbourhood search heuristic for solving the RCPSP. Given a partial schedule, i.e., a schedule which does not contain all activities, the neighbourhood for the heuristic is defined as the set of all schedules that also contain the partial schedule. In order to find the best schedule in the neighbourhood, Palpant et al. [2004] suggest an MILP model which finds optimal starting times for the missing activities w.r.t. the partial schedule. Note that the partial schedule is derived by removing activities from an initially complete schedule. The activities are removed according to different strategies.

Della Croce et al. [2014] use a similar approach as Palpant et al. [2004] for solving a single machine scheduling problem. The biggest difference between these contributions lies in the generation of the partial schedule. While Palpant et al. [2004] suggest different strategies for deriving a partial schedule, the algorithm of Della Croce et al. [2014] chooses a random position in a complete schedule and then removes, starting from the chosen position, a predetermined number of successive activities from the schedule.

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. The idea behind modes is to model different trade-offs between the processing time and the resource demands of an activity. An additional extension of MRCMPSPs is that it is also possible to consider multiple projects.

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 2 and 3 generate a schedule based on the assigned modes using CP. The last phase uses a large neighbourhood search to improve the schedule by changing the modes of some activities. Artigues and Hebrard [2013] use CP to find the optimal modes w.r.t. the specified neighbourhood. Phases 2 to 4 are repeated until the time limit is exceeded.

Toffolo et al. [2016] solve the MRCMPSP using a decomposition-based matheuristic. After fixing execution modes the problem is decomposed into time periods that are considered by independent MILP models. Finally, a hybrid local search is employed to improve the obtained solutions.

2.4 Time Window Discretization Models

Time discretization can be done in two ways. The first approach is to coarsen the time horizon in order to possibly obtain feasible but also less precise solutions, which are in general not optimal for the original problem. A different way of time discretization is to partition the given time horizon into subsets which, in contrast to the first approach, usually results in a relaxation of the original problem.

Early examples for time discretization by coarsening include Levin [1971] and Swersey and Ballard [1984]. The former deals with flight scheduling and routing problems. Departure times of aircrafts are represented as a bundle of time slots instead of continuous sets. Swersey and Ballard [1984] follow a similar approach for solving a bus scheduling problem.

An iterative refinement algorithm based on these ideas can be found in Boland et al. [2017] for solving the countinuous time service network design problem (CTSNDP). The authors solve the problem using a time-expanded network, in which each node represents a location and a time. 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.

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). The problem is formulated as a network flow model s.t. nodes of the graph correspond to time instants. Consequently, the formulation cannot cope with non integral travelling times. In such a case a relaxation of the original problem is derived by rounding the travelling times using special rounding procedures. In case the solution to the relaxation is not feasible for the original problem, the current time discretization is locally refined by disaggregating nodes of the current model.

A different way of time discretization is to partition the given time horizon into subsets. Such an approach is presented by Bigras et al. [2008] for a single machine scheduling problem. The scheduling horizon is partitioned into multiple sub periods. If a job spans several sub periods, the job gets split into multiple subjobs. The relaxation is solved via column generation. Each sub period with its corresponding jobs can be transferred into a subproblem for the used Dantzig-Wolfe decomposition (see Dantzig and Wolfe [1960]). The solution to the relaxation is then used as a lower bound in a B&B algorithm.

Other MILP approaches for solving single machine scheduling problems using time window discretization can be found in Baptiste and Sadykov [2009] and Boland et al. [2016]. Both contributions follow a common idea. By partitioning the given scheduling horizon, the number of variables in the MILP model decreases. In order to ensure the correctness of the model, additional constraints have to be added. Unlike Baptiste and Sadykov [2009], Boland et al. [2016] impose the additional restriction that a job spans at least two buckets.

An iterative refinement approach for the traveling salesman problem with time windows (TSPTW) can be found in Wang and Regan [2002] and Wang and Regan [2009]. 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 MILP model. As long as the gap between lower and upper bound is not sufficiently small, the scheduling horizon gets further refined and the problem is solved anew. In order to ensure that the overconstrained MILP model does not lead to worse solutions in subsequent iterations, the applied refinement scheme also takes the solution of the previous overconstrained MILP model into account.

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 relaxation.

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.

CHAPTER 3

Methods

In this chapter we discuss various theoretical foundations and optimization techniques upon which our algorithms are based from a theoretical point of view. First, we take a closer look at integer linear programming (ILP) and MILP models in general, as such a model constitutes the core of our algorithm. Afterwards, we review different heuristic techniques relevant to our algorithm. As mentioned before, our algorithm, consisting of an MILP component and a heuristic component, can be categorized as a matheuristic, which we discuss at the end of this chapter.

3.1 Mathematical Programming Methods

A mathematical programming problem deals with the task of finding a maximum or minimum value of a real valued function subject to a set of constraints. integer linear programming (ILP) is a subfield of mathematical programming as it focuses on linear objective functions and constraints only.

Many problems in computer science can be formulated as an ILP problem. While ILP alone is not sufficient to solve our problem in reasonable time for instances subject to a large time horizon, it constitutes an important part of our algorithm.

In the following, we first take a look at LP which is an “easy” variant of ILP, in the sense that LP problems can be solved in polynomial time. We review basic properties and the geometrical interpretation of LPs. Afterwards, we take a look at MILP. In contrast to LP, MILP problems are $\mathcal{NP}$-hard. Solving MILP problems is usually based on B&B. Hence, finding tight bounds on the optimal value of the problem's objective function is vital for an efficient B&B procedure. We will see that LP proves to be very useful for finding such bounds.

The review of mathematical programming is based on Bertsimas and Tsitsiklis [1997], Schrijver [1998] and Wolsey [1998].

3.1.1 Linear Programming

A linear programming (LP) problem is defined as follows:

$$\min \mathbf{c}'\mathbf{x} \tag{3.1}$$
$$\text{s.t. } \mathbf{a_i}'\mathbf{x} \geq b_i \qquad \forall i \in M_1 \tag{3.2}$$
$$\mathbf{a_i}'\mathbf{x} \leq b_i \qquad \forall i \in M_2 \tag{3.3}$$
$$\mathbf{a_i}'\mathbf{x} = b_i \qquad \forall i \in M_3 \tag{3.4}$$
$$x_j \geq 0 \qquad \forall j \in N_1 \tag{3.5}$$
$$x_j \leq 0 \qquad \forall j \in N_2 \tag{3.6}$$

The variables given by vector $\mathbf{x} = (x_1, \ldots, x_n)$ are called decision variables.

The goal of a linear program is to find a variable assignment $\mathbf{x}$ that minimizes the objective function (3.1) but does not violate any of the program's constraints (3.2) - (3.6).

If all constraints of the program are satisfied w.r.t. $\mathbf{x}$, then $\mathbf{x}$ is called a feasible solution. The set of all feasible solutions is called the feasible set or feasible region. Vector $\mathbf{x}$ is an optimal solution, if it is feasible and also minimizes the objective function. Note that more than one optimal solution may exist.

The set of all values that can be assigned to a decision variable x_j is called the domain of x_j. If the domain of x_j is restricted (see Constraints (3.5) - (3.6)), we refer to x_j as restricted. Otherwise x_j is called free or unrestricted.

The constraints of a linear program can be expressed as either equalities or inequalities. An equality constraint $\mathbf{a_i}'\mathbf{x} = b_i$ can be equivalently formulated with inequality constraints only: $\mathbf{a_i}'\mathbf{x} \leq b_i$ and $\mathbf{a_i}'\mathbf{x} \geq b_i$.

It is also possible to reverse the sign of the program's inequalities:

$$\mathbf{Ax} \leq \mathbf{b}$$
$$\Leftrightarrow -\mathbf{Ax} \geq -\mathbf{b}$$

Moreover, a minimization problem can be transformed into a maximization problem and vice versa:

$$\min \ \mathbf{c}'\mathbf{x} = \max \ -\mathbf{c}'\mathbf{x}$$

Therefore, we can write the above general form in a more compact way:

$$\min \mathbf{c}'\mathbf{x} \tag{3.7}$$
$$\text{s.t. } \mathbf{Ax} \geq \mathbf{b} \tag{3.8}$$
$$\mathbf{x} \in \mathbb{R}^n \tag{3.9}$$

Note that it is also possible to transform the Inequalities 3.9 into equalities by introducing slack variables $\mathbf{s}$:

$$\begin{aligned} &\mathbf{Ax} \geq \mathbf{b} \\ \Leftrightarrow &\mathbf{Ax} + \mathbf{s} = \mathbf{b} \qquad\qquad \mathbf{s} \geq \mathbf{0} \end{aligned}$$

Geometrical Interpretation of a Linear Program

Definition 1. *A polyhedron is a set that can be described in the form* $\{\mathbf{x} \in \mathbb{R}^n : \mathbf{Ax} \geq \mathbf{b}\}$*, where* $\mathbf{A}$ *is an* $m \times n$ *matrix and* $\mathbf{b}$ *is a vector in* $\mathbb{R}^m$*.*

The definition of a polyhedron bears strong similarities to the constraints of a linear program. In fact, a polyhedron describes the feasible region of a linear program. Moreover, for any linear program it holds that its corresponding polyhedron P is convex, i.e., if $\mathbf{x}, \mathbf{y} \in P$, then $\lambda\mathbf{x} + (1-\lambda)\mathbf{x} \in P$ for any $\lambda \in [0,1]$. It is easy to see that the optimal solution to a linear program has to be a corner point of the program's convex hull. Moreover, it turns out that an optimal solution to the linear program has to be an extreme point of P, i.e., a vector $\mathbf{x} \in P$ s.t. no two vectors $\mathbf{y}, \mathbf{z} \in P$ (different from $\mathbf{x}$) exist satisfying $\mathbf{x} = \lambda\mathbf{y} + (1-\lambda)\mathbf{z}$ for any $\lambda \in [0,1]$.

Solving Linear Programs

LP problems are $\mathcal{P}$-hard, i.e., they can be solved in polynomial time. There exist many different algorithms for solving LP problems. The first polynomial time algorithm for solving LP problems was the ellipsoid method, see Khachiyan [1980]. However, due to its poor performance in practice the ellipsoid method is only of theoretical interest. Other polynomial time algorithms are interior point methods, see Karmarkar [1984]. In contrast to the ellipsoid method, interior point methods are efficient in practice. One of the most effective methods is the simplex method by Dantzig [1951]. Although the simplex method has exponential worst case complexity, the algorithm is usually very fast in practice. The basic idea of the simplex method is to travel from one extreme point of the program's polyhedron to another extreme point along the edges of the polyhedron. If an extreme point is adjacent to more than one extreme point, the algorithm chooses the most cost reducing direction (w.r.t. minimization problems).

Note that there also exists polynomial time algorithms for solving LP problems (see Khachiyan [1979] and Karmarkar [1984]).

3.1.2 Mixed Integer Linear Programming

A mixed integer linear programming (MILP) problem is defined as follows:

$$\min \mathbf{c}'\mathbf{x} + \mathbf{d}'\mathbf{y} \tag{3.10}$$

$$\text{s.t. } \mathbf{Ax} + \mathbf{By} \leq \mathbf{b} \tag{3.11}$$

$$\mathbf{x}, \mathbf{y} \geq \mathbf{0} \tag{3.12}$$
$$\mathbf{x} \in \mathbb{Z}^n \tag{3.13}$$

MILP extends LP by allowing variables whose domains are restricted to the set of integers. If the program is based on integer variables only, we refer to the program as ILP. If the integer variables are additionally restricted to be either 0 or 1, the program is called binary integer linear program (BILP).

Solving MILP problems

In contrast to LP, MILP is $\mathcal{NP}$-hard (Papadimitriou [1981]). A basic procedure for solving an MILP problem is an algorithm which generates an increasing sequence of lower bounds (dual bounds)

$$\underline{\mathbf{x_1}} < \underline{\mathbf{x_2}} < \ldots < \underline{\mathbf{x_s}} \leq \mathbf{x}$$

and a decreasing sequence of upper bounds (primal bounds)

$$\overline{\mathbf{x_1}} > \overline{\mathbf{x_2}} > \ldots > \overline{\mathbf{x_t}} \geq \mathbf{x}$$

and terminates when

$$\underline{\mathbf{x_s}} - \overline{\mathbf{x_t}} \leq \epsilon$$

where ϵ is some small nonnegative value.

A primal bound is a lower bound for maximization problems and an upper bound for a minimization problems. Moreover, every feasible solution to an MILP problem is a primal bound.

A dual bound is a lower bound for minimization problems and an upper bound for maximization problems. Dual bounds are usually obtained by solving relaxations of the MILP problem.

Definition 2. *A problem* $z^{\mathrm{R}} = \min\{f(\mathbf{x}) : \mathbf{x} \in T \subseteq \mathbb{R}^n\}$ *is a relaxation of* $z = \min\{c(x) : \mathbf{x} \in X \subseteq \mathbb{R}^n\}$ *if:*

(i) $T \subseteq X$, *and*

(ii) $f(x) \leq c(x)\ \forall \mathbf{x} \in X$.

The idea of using a relaxation is to replace a difficult problem with a problem that is easier to solve. An MILP problem can for example be relaxed by discarding some of its constraints, which enlarges the set of feasible solutions. A common approach in this sense is the LP relaxation:

Definition 3. *For the MILP* $\min\{\mathbf{cx} : \mathbf{x} \in P \cap \mathbb{Z}^n\}$ *with* $P = \{x \in \mathbb{R}^n : \mathbf{Ax} \leq \mathbf{b}\}$, *the linear programming relaxation is the linear program* $z^{\mathrm{LP}} = \min\{\mathbf{cx} : \mathbf{x} \in P\}$.

For each problem there exists an ideal formulation z s.t. $z = z^{\mathrm{LP}}$. Such a formulation usually has a large number of constraints and is hard to find. However, in order to find an optimal solution, usually only a small amount of constraints is needed. The cutting plane method tries to utilize this fact and solves a given MILP formulation as follows: First, a relaxed version of the formulation is solved. If the solution of the relaxation is also a solution to the original MILP, then the solution is optimal. Otherwise, there exists at least one inequality of the MILP formulation that is violated. By adding these inequalities (cutting planes) to the relaxed formulation, the relaxation is strengthened and therefore provides a stronger lower bound. This procedure is repeated until an optimal solution is found. It is important to note that these added inequalities are required for obtaining a feasible solution. One could also use the same approach to add a set of strengthening inequalities which are not necessary for obtaining a feasible solution but may reduce the search space and hence speed up the solving process.

A very prominent procedure for solving MILPs is branch-and-bound (B&B) which divides the set of feasible solutions into subproblems and computes primal and dual bounds to decide whether a subproblem should be refined or discarded.

The cutting plane method can be embedded into a B&B procedure yielding the branch-and-cut (B&C) procedure. B&C usually generates cutting planes for each subproblem of the B&B tree, in order to generate stronger dual bounds for the subproblems.

Comparing Formulations

On the one hand, MILP is more expressive than LP, on the other hand, MILP problems are much harder to solve. A problem can be formulated in infinitely many (non equivalent) ways. MILP formulations can be compared by the polyhedra of their corresponding linear programming relaxation:

Definition 4. *Given a set $\mathbf{X} \in \mathbb{R}^n$ and two formulations P_1 and P_2 for $\mathbf{X}$, then*

(i) P_1 and P_2 are equivalent if $P_1 = P_2$,

(ii) P_1 is a stronger formulation than P_2 if $P_1 \subset P_2$, and

(iii) P_1 and P_2 are incomparable if $P_1 \not\subset P_2$ and $P_2 \not\subset P_2 1$.

3.2 Heuristics

There exist many problems for which exact methods are unsuitable, as they cannot solve the problem within reasonable time. Alternatively, one can resort to heuristic approaches for solving the problem. Heuristics focus on generating solutions of high quality, which can usually be found in significantly less computation time. However, they provide no dual bounds and therefore no quality guarantee on the computed solutions. In this chapter we review two basic heuristic concepts: construction heuristics and local search.

Afterwards we show how these two concepts can be combined to a new heuristic. We use Blum and Raidl [2016] as the basis of this review.

3.2.1 Construction Heuristics

Construction heuristics serve as basis for many other heuristic approaches. Starting from an empty solution, a construction heuristic iteratively expands the solution until it is complete. While the procedure is very fast, the generated solution usually leaves great room for improvement. A prominent example for a construction heuristic is a greedy heuristics, which chooses at each step of the solution generation the best element from a local point of view. Construction heuristics can also be randomized by simply choosing a random element to expand the current solution. The probability for an element to be chosen is usually weighted, depending on the impact the element has on the solution.

3.2.2 Local Search

In contrast to a construction heuristic, a local search procedure does not generate solutions from scratch. Instead, the goal of a local search procedure is to improve the quality of already existing solutions.

A local search procedure consists of three components. The first component is the neighbourhood function which assigns to a solution S a set of neighbours $N(S)$. Instead of explicitly defining the set of neighbours, a neighbourhood is usually defined by some (small) operation which, applied to S, generates all neighbours of S. The goal of local search is to find a local optimum, i.e., a solution S whose quality is not worse than any other solution in $N(S)$. Hence, a local optimum is a solution which is optimal w.r.t. some neighbourhood. A solution S can be improved by replacing it with a solution S' in $N(S)$ s.t. the quality of S' is higher than the quality of S. By repeating this procedure as long as possible one eventually reaches a local optimum.

The second local search component is the step function that decides which solution in $N(S)$ replaces the original solution S. One possibility is the so called first improvement method, which replaces S with the first found solution that has higher quality. Another way to replace S is the best improvement method, which replaces S with the solution that has the highest quality in $N(S)$. Moreover, the replacement for S can also be chosen randomly. Note that the choice of the most suitable step function is problem specific.

The last local search component is the termination criterion, which decides when to terminate the local search. A local optimum of a neighbourhood cannot always be found in reasonable time. Therefore, we prematurely terminate the local search if a specific criterion is met. A time limit is one of the most common termination criteria. However, the total number of iterations or the number of iterations without improvement are also popular choices.

Algorithm 3.1 shows a basic pseudocode for a local search.

Algorithm 3.1: Local Search

```
   Input: initial solution $S$
1: while $\exists S' \in N(S)$ s.t. $f(S') < f(S)$ and termination criteria not met do
2:     $S \leftarrow$ step function($N(S)$);
3: end while
4: return $S$;
```

Algorithm 3.2: GRASP

```
1: $S' \leftarrow S$; // stores the best found solution
2: while termination criteria not met do
3:     create a solution $S$ using a randomized construction heuristic;
4:     $S \leftarrow$ Local Search($S$);
5:     if $f(S) < f(S')$ then $S' \leftarrow S$;
6: end while
7: return $S'$;
```

3.2.3 GRASP

Metaheuristics are combinations of construction heuristics and/or local search procedures with other algorithms. The idea behind metaheuristics is to explore the search space more effectively than simple local search procedures. A greedy randomized adaptive search procedure (GRASP) is a prominent metaheuristic that applies a randomized variant of a construction heuristic followed by a local search component independently for many times, where the best found solution is kept as the result, see Resende and Ribeiro [2010]. Algorithm 3.2 shows a basic pseudocode for a GRASP algorithm.

3.3 Matheuristics

Matheuristics belong to the group of hybrid approaches. Hybrid approaches are usually a combination of two different algorithmic procedures. Matheuristics are a combination of mathematical programming and metaheuristics. The idea of matheursitics is to either improve the metaheuristic by exploiting mathematical programming techniques or improve the mathematical programming technique with the time efficiency of the metaheuristic (Caserta and Voß [2010]).

Matheuristics can be categorized in two types (Caserta and Voß [2010]). In the first type, a mathematical programming technique is embedded into a metaheuristic. For an example recall Palpant et al. [2004], who uses mathematical programming techniques in order to solve a large scale neighbourhood search heuristic for an RCPSP problem.

In the second type, the mathematical programming technique controls the calls to the metaheuristic. Typical applications for such matheuristics are MILP models which use heuristics to generate feasible solutions or dual bounds. To generate dual bounds, one can use the dual heuristic algorithm (Li et al. [2015]) mentioned in Section 2.2.

Feasible solutions can for example be generated by heuristics based on decomposition approaches of MILP models, e.g., the Lagrangian decomposition. The solution generated from a Lagrangian relaxation can in many cases be easily repaired s.t. the solution to the relaxation becomes feasible. Procedures to repair solutions are usually based on metaheuristics. For further examples of decomposition based heuristics see Raidl [2015].

CHAPTER 4

The Simplified Intraday Particle Therapy Patient Scheduling Problem

The simplified intraday particle therapy patient scheduling problem (SI-PTPSP) is defined on a set of activities $A = \{1, \ldots, \alpha\}$ and a set of unit-capacity resources $R = \{1, \ldots, \rho\}$. 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 $T = \{T^{\min}, \ldots, T^{\max}\}$ 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., $Y_a(t) = \{t, \ldots, t + p_a - 1\}$. 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 $L^{\min}_{a,a'} \leq L^{\max}_{a,a'}$, respectively. For each resource $r \in R$ a set of availability windows $W_r = \bigcup_{w=1,\ldots,\omega_r} W_{r,w}$ with $W_{r,w} = \{W^{\mathrm{start}}_{r,w}, \ldots, W^{\mathrm{end}}_{r,w}\} \subseteq T$ is given. Resource availability windows are non-overlapping and ordered according to starting time $W^{\mathrm{start}}_{r,w}$. Based on the resource availabilities and the precedence relations among the activities we can deduce for each activity a set of feasible starting times, denoted by $T_a \subseteq \{t^{\mathrm{r}}_a, \ldots, t^{\mathrm{d}}_a - p_a\}$; for details on the computation of this set see Section 6.1.

A feasible solution S (also called schedule) to the 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, i.e., a schedule with minimal total length.

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

4.1 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 by providing a reduction from the well known $\mathcal{NP}$-complete Partition problem.

Partition

INSTANCE: A finite set of n positive integers $B = \{b_0, b_1, \ldots, b_{n-1}\}$.
QUESTION: Can the set B be partitioned into two subsets B_1, B_2 s.t. the sum of the numbers in B_1 equals the sum of the numbers in B_2?

We adapt the aforementioned proof to show $\mathcal{NP}$-hardness of the SI-PTPSP. For this purpose we consider the decision problem variant of the SI-PTPSP, the k-SI-PTPSP:

k-SI-PTPSP

INSTANCE: An instance I of the SI-PTPSP and a non negative integer k.
QUESTION: Does there exist a solution to I with makespan less than or equal to k ?

Proposition 1. *The k-SI-PTPSP is $\mathcal{NP}$-complete.*

Proof. The proof consists of two parts. First, we show that k-SI-PTPSP is in $\mathcal{NP}$. Then, we show that k-SI-PTPSP is $\mathcal{NP}$-hard.

To show $\mathcal{NP}$-membership, consider the certificate relation $R = (I, S)$, where I is an instance of the k-SI-PTPSP and S is a schedule to I with makespan less than or equal to k. Since S is of size linear in I, it follows that R is polynomially balanced. Moreover, R is polynomially decidable as the schedule can be verified in $\mathcal{O}(|A|)$. Therefore, k-SI-PTPSP is in $\mathcal{NP}$.

$\mathcal{NP}$-hardness of k-SI-PTPSP is shown by a reduction from Partition. Consider an instance I of Partition as described above. Note that $\sum_{i=0}^{n-1} b_i$ has to be even, otherwise I cannot be a positive instance. Table 4.1 shows how to construct an instance I' of the k-SI-PTPSP from I.

Let (B_1, B_2) be a solution to I. Then, the following equation is valid:

$A = \{0, 1, \ldots, n\}$	$t^d_n = \frac{\sum_{i=0}^{n-1} b_i}{2} + 1$
$p_i = b_i \qquad \forall i \in \{0, \ldots, n-1\}$	$R = \{0\}$
$p_n = 1$	$W_0 = \{\{0, \ldots, T^{\max}\}\}$
$t^r_i = 0 \qquad \forall i \in \{0, \ldots, n-1\}$	$Q_i = \{0\} \qquad \forall i \in \{0, \ldots, n\}$
$t^r_n = \frac{\sum_{i=0}^{n-1} b_i}{2}$	$G = (A, \emptyset)$
$t^d_i = T^{\max} + 1 \qquad \forall i \in \{0, \ldots, n-1\}$	$T^{\max} = \sum_{i=0}^{n-1} b_i + 1$

Table 4.1: Rules for transforming an instance of Partition into an instance of the k-SI-PTPSP

$$\sum_{b_i \in B_1} b_i = \sum_{b_i \in B_2} b_i = t^r_n = T^{\max} - t^d_n = \frac{\sum_{i=0}^{n-1} b_i}{2} \tag{4.1}$$

Next, we show that I is a positive instance of Partition if I' is a positive instance of the k-SI-PTPSP. Let $S = \{S_0, \ldots, S_n\}$ be a solution to I'. Moreover, let, $B_1 = \{p_i : S_i < t^r_n\}$ and $B_2 = \{p_i : S_i \geq t^d_n\}$. Note that $B_1 \cup B_2 = B$. From Equation (4.1) it follows that $\sum_{b_i \in B_1} b_i = \sum_{b_i \in B_2} b_i$. Therefore, I is a positive instance of Partition.

It remains to show that I' is a positive instance of the k-SI-PTPSP if I is a positive instance of Partition: Let (B_1, B_2) be a solution to I. Moreover, let $A(B_i) = \{a_j : b_j \in B_i\}$ for $i \in \{1, 2\}$. W.l.o.g. assume that $A(B_1) = \{a_0, \ldots a_m\}$ and $A(B_2) = \{a_{m+1}, \ldots a_{n-1}\}$. Then, since Equation (4.1) is valid, the activities $a_i \in B_1$ can be scheduled at $S_i = \sum_{j=0}^{i-1} b_j$. Moreover, the activities $a_i \in B_2$ can be scheduled at $S_i = t^d_{p_n} + \sum_{j=m+1}^{i-1} b_j$. Let $S_n = t^r_n$. $\bigcup_{i=0}^{n} S_i$ is a solution to I', since $t^{\mathrm{r}}_i \leq S_i \leq t^{\mathrm{d}}_i - p_i$ for all $i \in \{0, \ldots, n\}$. Therefore, I' is a positive instance of the k-SI-PTPSP. $\square$

4.2 Mathematical Formulations

In this chapter we present various MILP models for the SI-PTPSP. We start by introducing two classical approaches: a discrete-event formulation (DEF) and a time-indexed formulation (TIF). Both serve as reference approaches to which we will compare our matheuristic. We show different time-indexed models and compare their strength. Afterwards, we present the time-bucket relaxation (TBR) formulation which is a relaxation of TIF and constitutes the central component of our matheuristic. We conclude the chapter by discussing additional inequalities for strengthening TBR.

4.2.1 Discrete Event Formulation

Discrete-event formulations (DEFs) are based on the idea of considering certain events that need to be ordered and for which respective times need to be found. 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 resource r, 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 modelled using variables S_a. Having transformed all resource events into activity events, the capacity of a resource now determines how many activities sharing a common resource can overlap in the schedule. As the capacity of all resources is one, no activities may overlap in the schedule. It suffices to check activity overlaps at events as resource requirements can only change there. For this purpose, we introduce variables $D_{r,k}$ which are one if resource $r \in R$ is used by any activity immediately after event k and zero otherwise. Variable MS denotes the makespan.

$$\min MS \tag{4.2}$$

$$S_a + p_a \leq MS \qquad \forall a \in A \tag{4.3}$$

$$S_{a'} - S_a \geq p_a + L^{\min}_{a,a'} \qquad \forall (a,a') \in P \tag{4.4}$$

$$S_{a'} - S_a \leq p_a + L^{\max}_{a,a'} \qquad \forall (a,a') \in P \tag{4.5}$$

$$\sum_{k \in K} x_{a,k} = 1 \qquad \forall a \in A' \tag{4.6}$$

$$\sum_{k \in K} y_{a,k} = 1 \qquad \forall a \in A' \tag{4.7}$$

$$\sum_{a \in A'} (x_{a,k} + y_{a,k}) = 1 \qquad \forall k \in K \tag{4.8}$$

$$E_{k-1} \leq E_k \qquad \forall k \in K \setminus \{1\} \tag{4.9}$$

$$E_k - M^{(4.10)}_{a,k}(1 - x_{a,k}) \leq S_a \qquad \forall k \in K, a \in A' \tag{4.10}$$

$$E_k + M^{(4.11)}_{a,k}(1 - x_{a,k}) \geq S_a \qquad \forall k \in K, a \in A' \tag{4.11}$$

$$E_k - M^{(4.12)}_{a,k}(1 - y_{a,k}) \leq S_a + p_a \qquad \forall k \in K, a \in A' \tag{4.12}$$

$$E_k + M^{(4.13)}_{a,k}(1 - y_{a,k}) \geq S_a + p_a \qquad \forall k \in K, a \in A' \tag{4.13}$$

$$D_{r,0} = \sum_{a \in A' : r \in Q_a} x_{a,0} \qquad \forall r \in R \tag{4.14}$$

$$D_{r,k} = D_{r,k-1} + \sum_{a \in A' : r \in Q_a} x_{a,k} - \sum_{a \in A' : r \in Q_a} y_{a,k} \qquad \forall k \in K \setminus \{1\}, r \in R \tag{4.15}$$

$$D_{r,k} \leq 1 \qquad \forall k \in K, r \in R \tag{4.16}$$

$$t^{\mathrm{r}}_a \leq S_a \leq t^{\mathrm{d}}_a - p_a \qquad a \in A' \tag{4.17}$$

$$MS, E_k, D_{r,k} \geq 0 \qquad \forall k \in K, r \in R \tag{4.18}$$

$$x_{a,k}, y_{a,k} \in \{0,1\} \qquad \forall k \in K, a \in A' \tag{4.19}$$

Inequalities (4.3) are used for determining the makespan. Precedence relations are enforced by Inequalities (4.4) and (4.5). According to Equalities (4.6) and (4.7) each activity starts and ends at precisely one event. Equalities (4.8) ensure that each event is assigned to either exactly one starting time or exactly one ending time of an activity. Events are chronologically ordered by Inequalities (4.9). Starting times of activities are linked to the corresponding start events by Inequalities (4.10) and (4.11). Similarly, Inequalities (4.12) and (4.13) link the event at which an activity a ends to the time at which the activity ends. We do not know in advance which event corresponds to which activity starting time. Hence, it is necessary to construct Inequalities (4.10) to (4.13) in such a way that they are valid for all feasible permutations of activities. This can be achieved by the so called big-M method, which puts events and activity starting times into relation w.r.t. a constant, usually large, offset M. The constraints are constructed in such a way that, M drops out of the constraint if an event coincides with an activity staring time. Otherwise, the offset M remains in the constraints in order to ensure that the constraints are valid. It is easy to find a high value for M s.t. the constraints are satisfied, e.g., $T^{\max}$. However, to make the LP relaxation as tight as possible, M should be as small as possible. In Section 6.2 we discuss how to find tight Big-M values for Constraints (4.10) to (4.13). Equalities (4.14) and (4.15) compute the total demand of a resource of all activities running during an event. Finally, Inequalities (4.16) ensure that all resource demands are met at all events. Inequalities (4.17) ensure that activities can only start during their release-time deadline windows. Inequalities (4.18) and (4.19) restrict the domains of the model's variables.

The formulation has $O(|A'|^2)$ variables and $O(|R| \cdot |A'|^2)$ constraints. Thus, DEF is a compact model, i.e., the model uses only one variable to represent an activity's starting time. However, its LP relaxation typically yields rather weak bounds primarily due to the inequalities involving the Big-M constants. Consequently, solving DEF to integrality frequently requires a huge number of B&B nodes and, thus, too much time. Our computational results in Chapter 7 will show that DEF is clearly not competitive with the other approaches we consider here.

4.2.2 Time-indexed Formulation

In a classical MILP way, we can model the 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 $t \in T_a$.

$$\min \; MS \tag{4.20}$$

$$\sum_{t \in T_a} x_{a,t} = 1 \qquad \forall a \in A \tag{4.21}$$

$$\sum_{t \in T_a} t \cdot x_{a,t} + p_a \leq MS \qquad \forall a \in A \tag{4.22}$$

$$\sum_{a \in A : r \in Q_a} \; \sum_{t' \in T_a : t \in Y_a(t')} x_{a,t'} \leq 1 \qquad \forall r \in R,\; t \in W_r \tag{4.23}$$

$$\sum_{t \in T_{a'}} t x_{a',t} - \sum_{t \in T_a} t x_{a,t} \geq p_a + L^{\min}_{a,a'} \qquad \forall (a,a') \in P \tag{4.24}$$

$$\sum_{t \in T_{a'}} t x_{a',t} - \sum_{t \in T_a} t x_{a,t} \leq p_a + L^{\max}_{a,a'} \qquad \forall (a,a') \in P \tag{4.25}$$

$$x_{a,t} \in \{0,1\} \qquad \forall a \in A,\; t \in T_a \tag{4.26}$$

$$MS \geq 0 \tag{4.27}$$

Equations (4.21) ensure that exactly one starting time is chosen for each activity. Inequalities (4.22) are used to determine the makespan *MS*. Resource restrictions are enforced by Inequalities (4.23). Last but not least, Constraints (4.24) and (4.25) guarantee that the precedence relations with their minimum and maximum time lags are respected. The remaining constraints specify the variable domains.

The model has $O(|A| \cdot |T|)$ variables and $O(|T| \cdot (|A| + |R| + |P|))$ constraints. Its size thus depends strongly on the resolution of the time discretization. Typically, the LP relaxation of TIF yields substantially tighter lower bounds than the LP relaxation of DEF, and thus less B&B nodes are usually required to solve TIF.

In the following we discuss two alternative stronger time-indexed formulations. Stronger formulations yield stronger LP relaxations, however their models are usually larger in size, which may make them harder to solve. For a comparison of different time-indexed models for a scheduling problem see Cavalcante et al. [2001].

Time-Indexed Formulation using Disaggregated Precedence Constraints

The precedence constraints of TIF can be disaggregated by replacing (4.24) with

$$\sum_{t' \in T_a : t' \leq t - p_a - L^{\min}_{a,a'}} x_{a,t'} \geq \sum_{t' \in T_{a'} : t' \leq t} x_{a',t'} \qquad \forall (a,a') \in P,\; t \in T_{a'} \tag{4.28}$$

and (4.25) with

$$\sum_{t' \in T_{a'}: t' \leq t + p_a + L^{\max}_{a,a'}} x_{a',t'} \geq \sum_{t' \in T_a : t' \leq t} x_{a,t'} \qquad \forall (a, a') \in P,\ t \in T_a \tag{4.29}$$

yielding the disaggregated time-indexed formulation (DTIF). Informally, constraints (4.28) enforce that if an activity a does not start after time t, then activity a' with $(a, a') \in P$ cannot start after time $t + p_a + L^{\max}_{a,a'}$. Similarly, constraints (4.29) enforce that if an activity a' does not start after time t, then activity a with $(a, a') \in P$ cannot start after time $t - p_a - L^{\min}_{a,a'}$. See Artigues [2013] for more details about disaggregated precedence constraints.

It is well known that time-indexed formulations using disaggregated precedence constraints are usually stronger formulations than their disaggregated equivalents (Artigues [2017]). Subsequently, we show that this also holds for TIF and DTIF.

Theorem 1. *The polyhedron of DTIF is a strict subset of the polyhedron of TIF.*

Proof. The proof consists of two parts. In the first part we show that DTIF is at least as strong as TIF. In the second part we show that the polyhedra are not isomorphic.

TIF differs from DTIF only by its minimum and maximum lag constraints. Hence, we have to show that Constraints (4.24) and (4.25) are implied by the constraints of DTIF.

For this purpose let $\tau_a \in T_a$ be the minimal time point for an activity $a \in A$ s.t.

$$\sum_{t \in T_a : t \leq \tau_a} x_{a,t} = 1 \qquad \forall a \in A \tag{4.30}$$

holds. Such a time point has to exist due to Inequalities (4.21). From Inequalities (4.28) it follows that

$$\sum_{t' \in T_a : t \leq \tau_a} x_{a,t'} \geq \sum_{t' \in T_{a'} : t' \leq \tau_a + p_a + L^{\min}_{a,a'}} x_{a',t'} \qquad \forall (a, a') \in P \tag{4.31}$$

and

$$\tau_a \leq \tau_{a'} \tag{4.32}$$

Now assume that $\sum_{t' \in T_{a'} : t' \leq \tau_a + p_a + L^{\min}_{a,a'}} x_{a',t'} = 0$. Then, due to Inequalities (4.21) and (4.32) it follows that

$$\sum_{t \in T_a} t \cdot x_{a,t} + p_a + L^{\min}_{a,a'} \leq \sum_{t \in T_{a'}} t \cdot x_{a',t} \qquad \forall (a, a') \in P \tag{4.33}$$

Hence, let us assume that $\sum_{t' \in T_{a'} : t' \leq \tau_a + p_a + L^{\min}_{a,a'}} x_{a',t'} = \sum_{t \in T_a : t \leq \tau_a} x_{a,t'} = 1$.

Then, the following inequalities can be derived:

$$\sum_{t \in T_a} t \cdot x_{a,t} + p_a + L^{\min}_{a,a'} = \sum_{t \in T_{a'}} t \cdot x_{a',t} \qquad \forall (a,a') \in P \tag{4.34}$$

Therefore, Inequalities (4.24) are implied by Inequalities (4.28) and (4.21). Analogously, it can be shown that Inequalities (4.25) are implied by Inequalities (4.29) and (4.21).

Next, we show that there exists a solution to the LP relaxation of TIF which is not a solution to the LP relaxation of DTIF. For this purpose consider the instance I described in Table (4.2).

$A = \{0,1\}$	$W_0 = \{[0, T^{\max}]\}$
$p_i = 1 \quad \forall i \in \{0,1\}$	$Q_i = 0 \quad \forall i \in \{0,1\}$
$t^r_i = 0 \quad \forall i \in \{0,1\}$	$G = (A, \{(0,1)\})$
$t^d_i = 6 \quad \forall i \in \{0,1\}$	$L^{\min}_{0,1} = 0$
$T^{\max} = 5$	$L^{\max}_{0,1} = T^{\max}$
$R = \{0\}$	

Table 4.2: An instance of the SI-PTPSP for which a solution to the LP relaxation of TIF exists, but no solution to the LP relaxation of DTIF.

One can easily verify that the following is an LP-solution to I w.r.t. TIF:

$x_{0,3} = 1$

$x_{1,1} = 0.1$

$x_{1,5} = 0.9$

$MS = 5.6$

However, Inequalities (4.28) are clearly violated for $t = 3$ since $\sum_{t\in\{1,2\}} x_{0,t} = 0$, $\sum_{t\in\{1,\dots,3\}} x_{1,t} = 0.1$ and $0 \ngeq 0.1$ Hence, the solution is not an LP-solution w.r.t. DTIF.

□

Time-Indexed Formulation based on Step Variables

It is also possible to model the SI-PTPSP by the following time-indexed formulation with step variables (STIF) using binary variables $y_{a,t}$ for indicating that activity $a \in A$ has started at some time $t' \leq t \in T_a$, and thus is finished at time $t + p_a$. While the polyhedra of STIF and DTIF are isomorphic (see 4.2.3), STIF may have a computational advantage over DTIF w.r.t. B&B as branching on the variables of STIF may divide the search space more evenly, than branching on the variables of DTIF (see Cavalcante et al. [2001]). More details about time-indexed models based on step variables can be found in Artigues [2013].

Let $T^{\min}_a = \min(T_a)$, $T^{\max}_a = \max(T_a)$ and

$$\text{pred}_a(t) = \max\{t' \in T_a : t' < t\} \quad \text{and} \quad \text{succ}_a(t) = \min\{t' \in T_a : t' > t\} \tag{4.35}$$

indicate the predecessor and successor time points existing in T_a for some $T_a^{\min} < t$ and $t < T_a^{\max}$, respectively.

Furthermore, let

$$\xi_{a,t} = \begin{cases} y_{a,t} & \text{for } t \in T_a, \\ 0 & \text{for } t < T_a^{\min}, \\ 1 & \text{for } t > T_a^{\max}, \\ y_{a,\mathrm{pred}_a(t)} & \text{else} \end{cases} \tag{4.36}$$

be a generalization of the variables $y_{a,t}$ that is safely defined for any $t \in \mathbb{Z}$.

We can now define STIF as follows.

$$\min \mathit{MS} \tag{4.37}$$

$$T_a^{\max} - \left(\sum_{t \in T_a \setminus \{T_a^{\max}\}} (\mathrm{succ}_a(t) - t) \cdot y_{a,t} \right) + p_a \le \mathit{MS} \qquad \forall a \in A \tag{4.38}$$

$$\sum_{a \in A : (r \in Q_a \wedge t \in \{T_a^{\min}, \ldots, T_a^{\max} + p_a\})} \xi_{a,t} - \xi_{a,t-p_a} \le 1 \qquad \forall r \in R,\ t \in W_r \tag{4.39}$$

$$\xi_{a,t-p_a-L_{a,a'}^{\min}} \ge y_{a',t} \qquad \forall (a,a') \in P, t \in T_{a'} \tag{4.40}$$

$$\xi_{a,t-p_a-L_{a,a'}^{\max}} \le y_{a',t} \qquad \forall (a,a') \in P, t \in T_{a'} \tag{4.41}$$

$$y_{a,t} \le y_{a,\mathrm{succ}_a(t)} \qquad \forall a \in A,\ t \in T_a \setminus \{T_a^{\max}\} \tag{4.42}$$

$$y_{a,T_a^{\max}} = 1 \qquad \forall a \in A \tag{4.43}$$

$$y_{a,t} \in \{0,1\} \qquad \forall a \in A,\ t \in T_a \setminus \{T_a^{\max}\} \tag{4.44}$$

$$\mathit{MS} \ge 0 \tag{4.45}$$

Inequalities (4.38) are used to determine the makespan MS. They essentially count the number of time slots at which each activity $a \in A$ has not yet started. Note that $y_{a,T_a^{\max}}$ must always be one (also cf. (4.43)) and therefore is omitted in the sum. Resource restrictions are enforced for each time slot in which a resource $r \in R$ is available by Inequalities (4.39). Constraints (4.40) and (4.41) guarantee that the precedence relations with their minimum and maximum time lags are respected. Last but not least, Inequalities (4.42) and (4.43) ensure that for each $a \in A$ the sequence $y_{a,T^{\min}}, \ldots, y_{a,T^{\max}}$ never decreases and ends with one, i.e., the activity is actually started at some time.

Proposition 2. *Let $(\mathbf{y}, \mathit{MS})$ be a solution to STIF. Then, the vector S with the values $S_a = T_a^{\max} - \sum_{t \in T_a \setminus \{T_a^{\max}\}} (\mathrm{succ}_a(t) - t) \cdot y_{a,t}$ for all $a \in A$ is a solution to the SI-PTPSP.*

4.2.3 Comparison of STIF and DTIF

Consider the following bijection between the variables of STIF and the variables of DTIF:

$$\xi_{a,t} = \sum_{t' \in T_a : t' \leq t} x_{a,t'} \qquad \forall a \in A, t \in \mathbb{Z} \tag{4.46}$$

Equation (4.46) can be reformulated as

$$\begin{aligned} \xi_{a,t} &= x_{a,t} + \sum_{t' \in T_a : t' \leq t-1} x_{a,t'} \\ &= x_{a,t} + \xi_{a,t-1} \end{aligned}$$

yielding a new identity

$$x_{a,t} = \xi_{a,t} - \xi_{a,t-1} \tag{4.47}$$

Theorem 2. *The polyhedra of STIF and DTIF are isomorphic.*

Proof. We prove Theorem 2 by showing that the constraints of STIF are a transformation of the constraints of DTIF and vice versa. The variables of the constraints can be transformed by Identities (4.46) and (4.47).

First, we show how Inequalities (4.43) can be transformed into Inequalities (4.21) and vice versa using Equation (4.46):

$$y_{a,T_a^{\max}} = \xi_{a,T_a^{\max}} = \sum_{t' \in T_a : t' \leq T_a^{\max}} x_{a,t'} = \sum_{t' \in T_a} x_{a,t'} = 1 \qquad \forall a \in A$$

Next, we show the transformation between Inequalities (4.22) and Inequalities (4.38):

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

Substituting according to Equation (4.47) on the left-hand side of the inequality yields:

$$\sum_{t \in T_a} t \cdot x_{a,t} = \sum_{t \in T_a} t \cdot (\xi_{a,t} - \xi_{a,t-1})$$

Note that $\xi_{a,t-1} = y_{a,\mathrm{pred}_a(t)}$ for all $t \in T_a \wedge t > T_a^{\min}$ and $\xi_{a,T_a^{\min}-1} = 0$. Let $T_a = \{t_1, \ldots, t_k\}$ with $t_1 = T_a^{\min}$ and $t_k = T_a^{\max}$. Then, it follows that

$$\begin{aligned} &\sum_{t \in T_a} t \cdot (\xi_{a,t} - \xi_{a,t-1}) = T_a^{\min} \cdot y_{a,T_a^{\min}} + \sum_{t \in T_a \setminus T_a^{\min}} t \cdot (y_{a,t} - y_{a,\mathrm{pred}_a(t)}) \\ &= t_1 \cdot y_{a,t_1} + t_2 \cdot (y_{a,t_2} - y_{a,t_1}) + t_3 \cdot (y_{a,t_3} - y_{a,t_2}) + \ldots + t_k \cdot (y_{a,t_k} - y_{a,t_{k-1}}) \\ &= y_{a,t_1} \cdot (t_1 - t_2) + y_{a,t_2} \cdot (t_2 - t_3) + \ldots + y_{a,t_{k-1}} \cdot (t_{k-1} - t_k) + t_k \cdot y_{a,t_k} \\ &= \sum_{t \in T_a \setminus T^{\max}} (t - \mathrm{succ}_a(t)) \cdot y_{a,t} + T^{\max} \cdot y_{a,T^{\max}} \end{aligned}$$

Hence,

$$\sum_{t \in T_a} t \cdot x_{a,t} = T^{\max} - \sum_{t \in T_a \setminus T^{\max}} (\mathrm{succ}_a(t) - t) \cdot y_{a,t}$$

as desired.

Next, we focus on transforming the resource constraints of STIF and DTIF. For this purpose, we prove the following identity for arbitrary t:

$$\xi_{a,t} - \xi_{a,t-p_a} = \sum_{t' \in T_a : t \in Y_a(t')} x_{a,t'}$$

We first reformulate the right-hand side of the equation as the difference of two sums and then show that each sum corresponds to a ξ term of the left-hand side of the identity.

From the definition of $Y_a(t')$ it follows that $t - p_a + 1 \leq t' \leq t$. Let t'_1 denote the earliest time point s.t. $t'_1 \in T_a \wedge t - p_a + 1 \leq t'_1$. Equivalently, we define t'_k as the latest time point s.t. $t'_k \in T_a \wedge t'_k \leq t$. Then,

$$\sum_{t' \in T_a : t \in Y_a(t')} x_{a,t'} = \sum_{t' \in T_a : t' \leq t'_k} x_{a,t'} - \sum_{t' \in T_a : t' < t'_1} x_{a,t'}$$

From the definition of t'_1 and t'_k it follows that

$$\sum_{t' \in T_a : t' \leq t'_k} x_{a,t'} = \xi_{a,t'_k} = \xi_{a,t}$$
$$\sum_{t' \in T_a : t' < t'_1} x_{a,t'} = \xi_{a,t'_1} = \xi_{a,\mathrm{pred}_a(t-p_a+1)} = \xi_{a,t-p_a}$$

Therefore, the identity holds. By applying the identity on Constraints (4.23), we get

$$\sum_{a \in A : r \in Q_a} \xi_{a,t} - \xi_{a,t-p_a} \leq 1 \qquad \forall r \in R, t \in W_r$$

which is equivalent to Constraints (4.39) due to the definition of the ξ variables.

Moreover, applying the identity on Constraints (4.39), yields

$$\sum_{a \in A : (r \in Q_a \wedge t \in \{T_a^{\min}, \ldots, T_a^{\max} + p_a\})} \sum_{t' \in T_a : t \in Y_a(t')} x_{a,t'} \leq 1 \qquad \forall r \in R,\ t \in W_r$$

which corresponds to Constraints (4.23) since $t' \in T_a : t \in Y_a(t')$ implies that $t \in \{T_a^{\min}, \ldots, T_a^{\max} + p_a\}$.

Next, we show the transformation between the minimum time lag Constraints (4.28) and (4.40). Consider Inequalities (4.28):

$$\sum_{t' \in T_a : t' \leq t - p_a - L_{a,a'}^{\min}} x_{a,t'} \geq \sum_{t' \in T_{a'} : t' \leq t} x_{a',t'} \qquad \forall (a, a') \in P,\ t \in T_{a'} \tag{4.48}$$

By Equation (4.46) it must hold that

$$\sum_{t' \in T_a : t' \le t - p_a - L^{\min}_{a,a'}} x_{a,t'} = \xi_{a,t-p_a-L^{\min}_{a,a'}} \qquad \text{and}$$

$$\sum_{t' \in T_{a'} : t' \le t} x_{a',t'} = y_{a',t}$$

Therefore, the transformation holds. The maximum lag constraints of the models can be transformed analogously.

Inequalities (4.42) of STIF do not have an equivalent in DTIF but we can show that they are implied. By substituting according to Equation (4.46), Inequalities (4.42) are transformed to:

$$y_{a,t} = \sum_{t' \in T_a : t' \le t} x_{a,t} \tag{4.49}$$

$$y_{a,\mathrm{succ}_a(t)} = \sum_{t' \in T_a : t' \le succ_a(t)} x_{a,t} \tag{4.50}$$

Therefore, the transformation yields

$$\sum_{t' \in T_a : t' \le t} x_{a,t} \le \sum_{t' \in T_a : t' \le succ_a(t)} x_{a,t} \qquad \forall a \in A, t \in T_a \setminus \{T^{\max}\} \tag{4.51}$$

As $t < \mathrm{succ}_a(t)$, it follows that $\sum_{t' \in T_a : t' \le t} x_{a,t}$ is contained in $\sum_{t' \in T_a : t' \le succ_a(t)} x_{a,t}$. Therefore, the transformation yields an inequality which always holds.

$\square$

4.2.4 Time-bucket Relaxation

As the number of variables and constraints of TIF can become huge 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 lower bound to the optimal value of the original problem but in general not directly a valid solution to it. Note that this stays in contrast to a more common approach in which the time-discretization is coarsened in order to obtain a feasible but also less precise solution, which is not necessarily optimal (or even feasible) for the original problem. Based on TBR we will build our iterative refinement approach in the subsequent chapter that is guaranteed to converge to an optimal solution for SI-PTPSP.

Let $B = \{B_1, \ldots, B_\beta\}$ be such 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 $B_b = \{B^{\mathrm{start}}_b, \ldots, B^{\mathrm{end}}_b\}$ contained in the bucket. Since B is a partitioning of T we have

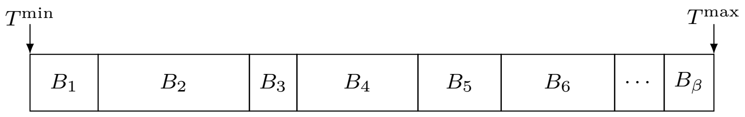


Figure 4.1: Bucket partitioning of T.

$B_1^{\text{start}} = T^{\min}$, $B_\beta^{\text{end}} = T^{\max}$, and $B_b^{\text{end}} + 1 = B_{b+1}^{\text{start}}$, $\forall b \in I(B) \setminus \{\beta\}$. For an illustration see Figure 4.1. 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 possibly 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 $C_a = \{C_{a,1}, \dots, C_{a,\gamma_a}\} \subseteq 2^{I(B)}$. Let functions $\text{bfirst}(a,c)$ and $\text{blast}(a,c)$ for $a \in A$ and $c = 1, \dots, \gamma_a$ 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 according to $(\text{bfirst}(a,c), \text{blast}(a,c))$. We can determine all bucket sequences for an activity in time $O(|T|)$ by "sliding" the activity over all time slots and taking the covered buckets. With a more careful approach this can be brought down to run in $O(|B| \log |B|)$, see Section 6.3 on how this is done in detail. 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 possibly 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 $z_{a,b,c}^{\min} = z_{a,b,c}^{\max} = |B_b|$.

Figure 4.2 shows an example of 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 completely performed in bucket sequence $C_{a,c}$ for $c \in 1, \dots, \gamma_a$. Model TBR is stated as follows:

$$\min \; MS \tag{4.52}$$

$$\sum_{c=0}^{\gamma_a - 1} y_{a,c} = 1 \qquad \forall a \in A \tag{4.53}$$

$$\sum_{c=0}^{\gamma_a - 1} S_{a,c}^{\min} \cdot y_{a,c} + p_a \leq MS \qquad \forall a \in A \tag{4.54}$$

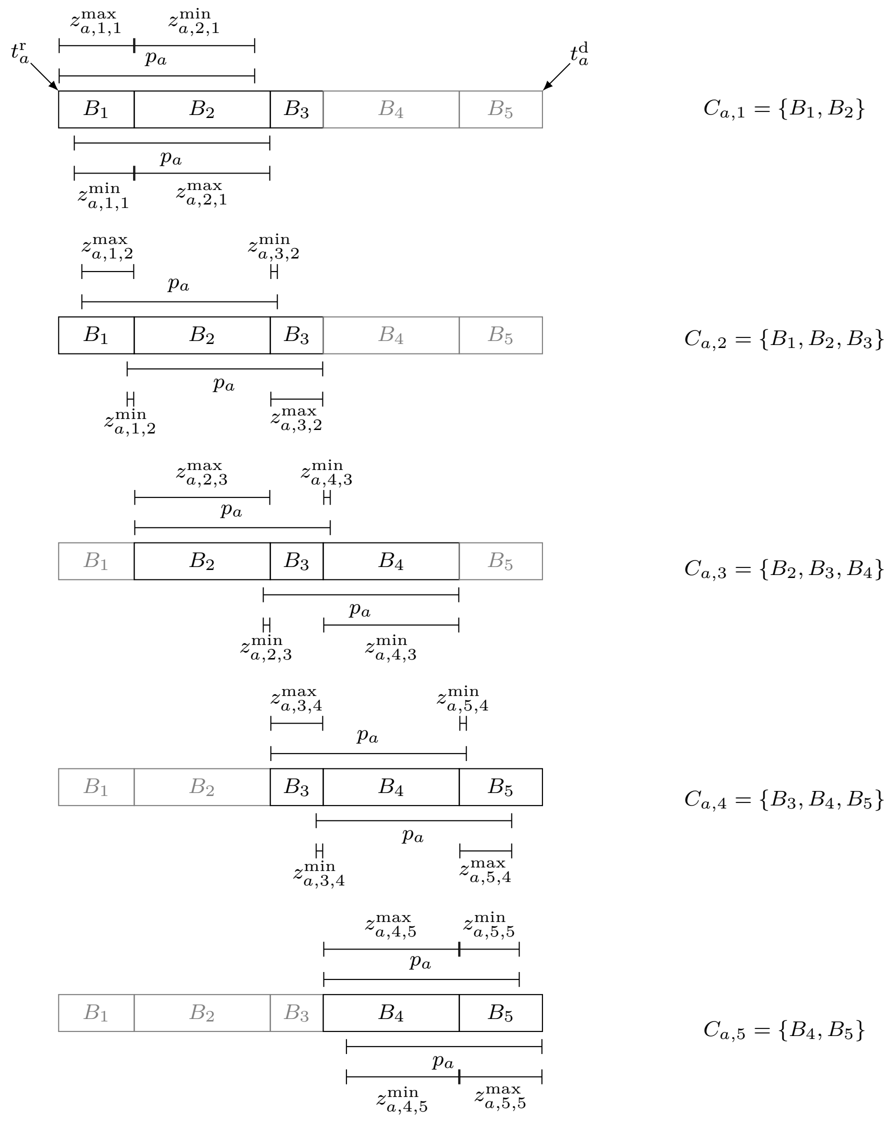


Figure 4.2: Bucket sequences C_a of an activity a with processing time p_a. Descriptions of inner buckets of a sequence are omitted since we always have $z^{\min}_{a,b,c} = z^{\max}_{a,b,c} = |B_b|$ for them.

$$\sum_{a \in A: r \in Q_a} \sum_{C_{a,c} \in C_a : b \in C_{a,c}} z^{\min}_{a,b,c} \cdot y_{a,c} \leq W^B_r(b) \qquad \forall r \in R,\ b \in I(B) \tag{4.55}$$

$$\sum_{c'=1}^{\gamma_{a'}} S^{\max}_{a',c'} \cdot y_{a',c'} - \sum_{c=1}^{\gamma_a} S^{\min}_{a,c} \cdot y_{a,c} \geq p_a + L^{\min}_{a,a'} \qquad \forall (a,a') \in P \tag{4.56}$$

$$\sum_{c'=1}^{\gamma_{a'}} S^{\min}_{a',c'} \cdot y_{a',c'} - \sum_{c=1}^{\gamma_a} S^{\max}_{a,c} \cdot y_{a,c} \leq p_a + L^{\max}_{a,a'} \qquad \forall (a,a') \in P \tag{4.57}$$

$$y_{a,c} \in \{0,1\} \qquad \forall a \in A,\ c = 1,\ldots,\gamma_a \tag{4.58}$$

$$MS \geq 0 \tag{4.59}$$

Equations (4.53) ensure that exactly one bucket sequence is chosen for each activity. The makespan MS is determined using Inequalities (4.54). Constraints (4.55) 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 (4.56) and (4.57) realize the precedence constraints with their minimum and maximum time lags, respectively. These restrictions are also 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 directly depend on $|T|$.

Similar to TIF, we can disaggregate the precedence constraints by replacing (4.56) by

$$\sum_{c'=1,\ldots,\gamma_a : S^{\min}_{a,c'} \leq S^{\max}_{a',c} - p_a - L^{\min}_{a,a'}} y_{a,c'} \geq \sum_{c'=1,\ldots,c} y_{a',c'} \qquad \forall (a,a') \in P,\ c = 1,\ldots,\gamma_{a'} \tag{4.60}$$

and (4.57) by

$$\sum_{c'=1,\ldots,\gamma_{a'} : S^{\min}_{a',c'} \leq S^{\max}_{a,c} + p_a + L^{\max}_{a,a'}} y_{a',c'} \geq \sum_{c'=1,\ldots,c} y_{a,c'} \qquad \forall (a,a') \in P,\ c = 1,\ldots,\gamma_a \tag{4.61}$$

yielding the disaggregated time-bucket relaxation (DTBR).

Extended Time-bucket Relaxation

We originally started with a more elaborate TBR formulation documented here. This formulation, however, turned out to not perform well in practice due to its substantially larger number of variables and constraints.

We strengthen the TBR by guaranteeing that the total execution time of an activity spent across all buckets equals its processing time. To this end we introduce additional variables $z_{a,b} \geq 0$ indicating the number of time-slots, activity $a \in A$ is performed within bucket $b \in I(B,a)$.

Set $I(B,a) = \bigcup_{c=1,\ldots,\gamma_1} C_{a,c}$ refers to the set of all buckets in which a part of activity $a \in A$ may possibly be performed.

The extended time-bucket relaxation (ETBR) is stated as follows:

$$\min MS \tag{4.62}$$

$$\sum_{c=1}^{\gamma_a} y_{a,c} = 1 \qquad \forall a \in A \tag{4.63}$$

$$\sum_{c=1}^{\gamma_a} S^{\min}_{a,c} \cdot y_{a,c} + p_a \leq MS \qquad \forall a \in A \tag{4.64}$$

$$\sum_{c'=1}^{\gamma_{a'}} S^{\max}_{a',c'} \cdot y_{a',c'} - \sum_{c=1}^{\gamma_a} S^{\min}_{a,c} \cdot y_{a,c} \geq p_a + L^{\min}_{a,a'} \qquad \forall (a,a') \in P \tag{4.65}$$

$$\sum_{c'=1}^{\gamma_{a'}} S^{\min}_{a',c'} \cdot y_{a',c'} - \sum_{c=1}^{\gamma_a} S^{\max}_{a,c} \cdot y_{a,c} \leq p_a + L^{\max}_{a,a'} \qquad \forall (a,a') \in P \tag{4.66}$$

$$B^{\text{start}}_{\text{blast}(a,c)} \cdot y_{a,c} + z_{a,\text{blast}(a,c)} \leq MS \qquad \forall a, \in A, b \in I(B,a) \tag{4.67}$$

$$z_{a,b} \geq \sum_{c=1,\ldots,\gamma_a : b \in C_{a,c}} z^{\min}_{a,b,c} \cdot y_{a,c} \qquad \forall a \in A,\ b \in I(B,a) \tag{4.68}$$

$$z_{a,b} \leq \sum_{c=1,\ldots,\gamma_a : b \in C_{a,c}} z^{\max}_{a,b,c} \cdot y_{a,c} \qquad \forall a \in A,\ b \in I(B,a) \tag{4.69}$$

$$\sum_{b \in I(B,a)} z_{a,b} = p_a \qquad \forall a \in A \tag{4.70}$$

$$\sum_{a : b \in I(B,a) \wedge r \in Q_a} z_{a,b} \leq W^B_r(b) \qquad \forall r \in R,\ b \in I(B) \tag{4.71}$$

$$0 \leq z_{a,b} \leq \max_{c : b \in C_{a,c}} z^{\max}_{a,b,c} \qquad \forall a \in A,\ b \in I(B,a) \tag{4.72}$$

$$y_{a,c} \in \{0,1\} \qquad \forall a \in A,\ c = 1,\ldots,\gamma_a \tag{4.73}$$

$$MS \geq 0 \tag{4.74}$$

Inequalities (4.63)–(4.67),(4.72)–(4.74) correspond to Inequalities (4.52)–(4.54), (4.56)–(4.59) of TBR. Inequalities (4.68) and (4.69) link the $z_{a,b}$ variables and the $y_{a,c}$ variables. Equations (4.70) ensure that each activity's whole processing time is allocated. Inequalities (4.71) consider the resource availabilities for performing the activities.

The ETBR model has $\mathcal{O}((|A| + |R|) \cdot |B| + |P|)$ constraints, while TBR only has $\mathcal{O}(|A| + |R| \cdot |B| + |P|)$ constraints. Hence, for a high number of activities the number of constraints drastically increases. Preliminary tests have shown that the additional constraints slow down the model significantly compared to TBR. Moreover, the accuracy gained by introducing the z variables is only very small, especially for a coarse bucket partitioning. Therefore, the costs of the ETBR model outweigh its benefits. Note that Inequalities (4.67)–(4.69) add $\mathcal{O}(|A| \cdot |B|)$ constraints to the model.

4.2.5 Comparison of TIF and TBR

In this section we compare TIF and TBR. We first show that the LP relaxations of TIF and TBR are equally strong if all buckets of TBR have unit size. Afterwards, we prove that TBR is a relaxation of TIF for an arbitrary bucket partitioning of 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 $T_a = \{S^{\min}_{a,c} : C_{a,c} \in \mathcal{C}_a\} = \{S^{\max}_{a,c} : C_{a,c} \in \mathcal{C}_a\}$ and $|T_a| = |\mathcal{C}_a|$ for all $a \in A$. Moreover, since buckets correspond to time points in this scenario, resource availabilities become binary per bucket.

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

Proposition 3. *Function φ is bijective.*

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

Proposition 4. *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 (4.21) into (4.53), (4.22) into (4.54), (4.24) into (4.56), and (4.25) into (4.57) and vice versa. To provide the isomorphism between (4.23) and (4.55) we need a few further things. First recall that all $z^{\min}_{a,b,c}$ constants are equal to 1. Secondly, 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, this 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.*

In the following we show that TBR with an arbitrary bucket partitioning is a relaxation of TBR$_1$ and thus of TIF.

Definition 5. *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 $\forall b' \in B' \, \exists b \in B (b' \subseteq b)$.*

Definition 6. *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 $C'_{a,c'} \in C'_a$:*

$$\bigcup_{b' \in C'_{a,c'}} b' \subseteq \bigcup_{b \in \sigma(C'_{a,c'})} b \wedge \forall C_{a,c_i} \in C_a \left(\bigcup_{b' \in C'_{a,c'}} b' \nsubseteq \bigcup_{b \in C_{a,c_i}} b \vee \sigma(C'_{a,c'}) \subseteq C_{a,c_i} \right)$$

This means sigma 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:*

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

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 $S^{\min}_{a,c'} \in \operatorname{bfirst}(a,c)$ and $(S^{\min}_{a,c'} + p_a) \in \operatorname{blast}(a,c)$.

Bucket sequences $C'_{a,c'}$ and $C_{a,c}$ satisfy $\bigcup_{b' \in C'_{a,c'}} b' \subseteq \bigcup_{b \in C_{a,c}} b$. 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 3. *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:

$$y_{a,c} = \begin{cases} 1 & \exists C'_{a,c'} \in C'_a (y^*_{a,c'} = 1 \wedge \sigma(C'_{a,c'}) = C_{a,c}) \\ 0 & \text{otherwise} \end{cases} \tag{4.75}$$

We first show that y is a feasible solution to TBR_B. Constraints (4.53) are satisfied since $y^*_{a,c'}$ is feasible and σ is surjective. As $\operatorname{bfirst}(a,c') \subseteq \operatorname{bfirst}(a,c)$ 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 (4.54),(4.56), and (4.57) must hold. If Inequalities (4.55) 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 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.*

4.2.6 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.

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].

The respective constraints are specified in terms of the following sets. First we determine for each $b \in I(B)$ sets $\mathcal{S}_b = \{(a,c) : a \in A, c \in C_a, z^{\min}_{a,b,c} < |B_b|, |C_{a,c}| > 1, \mathrm{bfirst}(a,c) = b\}$ and $\mathcal{F}_b = \{(a,c) : a \in A, c \in C_a, z^{\min}_{a,b,c} < |B_b|, |C_{a,c}| > 1, \mathrm{blast}(a,c) = b\}$ of non-unit bucket sequences starting and ending in bucket B_b, respectively. For each of these sets we consider a graph having the respective set as vertices and an edge between two vertices 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. Then, we add the following inequalities to TBR:

$$\sum_{(a,c)\in\kappa} y_{a,c} \le 1, \qquad \forall b \in I(B), \forall \kappa \in \mathcal{C}^{\mathcal{S}}_b \tag{4.76}$$

$$\sum_{(a,c)\in\kappa} y_{a,c} \le 1, \qquad \forall b \in I(B), \forall \kappa \in \mathcal{C}^{\mathcal{F}}_b \tag{4.77}$$

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 (4.55). 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 vertices. 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 $\mathcal{S}_{b,r} = \mathcal{S}_b \cap \{(a,c) : a \in A, c \in C_a, r \in Q_a\}$ of $\mathcal{S}_b$ and subsets $\mathcal{F}_{b,r} = \mathcal{F}_b \cap \{(a,c) : a \in A, c \in C_a, r \in Q_a\}$ 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 build the same type of constraints:

$$\sum_{(a,c)\in\mathcal{S}_{b,r}} y_{a,c} \leq 1, \qquad \forall b \in I(B), \forall r \in R : |\mathcal{S}_{b,r}| \geq 2 \tag{4.78}$$

$$\sum_{(a,c)\in\mathcal{F}_{b,r}} y_{a,c} \leq 1, \qquad \forall b \in I(B), \forall r \in R : |\mathcal{F}_{b,r}| \geq 2 \tag{4.79}$$

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.

Path Inequalities

The idea of this kind of inequalities is to extend the precedence constraints (4.56) and (4.57) and the makespan constraints (4.59) 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 $\pi_{a_0,a_m} = (a_0, \ldots, a_m)$ be a directed path from activity a_0 to activity a_m in G. Moreover, let $d_{L^{\min}}(\pi_{a_0,a_m}) = \sum_{i=0}^{m-1} p_{a_i} + L^{\min}_{a_i,a_{i+1}}$ 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 $a, a' \in A$. Let $\Pi = \bigcup_{\{a,a'\}\subseteq A} \Pi_{a,a'}$ denote the union of all these paths between any two 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:

$$\sum_{c=1}^{\gamma_a} S^{\min}_{a,c} \cdot y_{a,c} + d_{L^{\min}}(\pi_{a,a'}) \leq \sum_{c'=1}^{\gamma_{a'}} S^{\max}_{a',c'} \cdot y_{a',c'} \tag{4.80}$$

$$\sum_{c=1}^{\gamma_a} S^{\max}_{a,c} \cdot y_{a,c} + d_{L^{\max}}(\pi_{a,a'}) \geq \sum_{c'=1}^{\gamma_{a'}} S^{\min}_{a',c'} \cdot y_{a',c'} \tag{4.81}$$

$$\sum_{c=1}^{\gamma_a} S^{\min}_{a,c} \cdot y_{a,c} + d_{L^{\min}}(\pi_{a,a'}) + p_{a'} \leq MS \tag{4.82}$$

Due to the exponential number of these inequalities we only consider a reasonable subset of them in our implementation, for details see Section 6.4.

CHAPTER 5

Iterative Time-Bucket Refinement Algorithm

Solving instances of the SI-PTPSP using a MILP formulation is possible in theory, however in practice this approach may suffer from the disadvantages of the MILP formulation in use, especially for instances with a large time horizon. While DEF is expected to yield weak LP relaxations, the performance of TIF (and TBR) strongly depends on the size of the time horizon as it is proportional to the number of variables in the model.

The basic idea of our iterative time-bucket refinement algorithm (ITBRA) is to solve instances of the SI-PTPSP without explicitly considering the complete time horizon of the instance. Solving an instance of SI-PTPSP using TBR w.r.t. a coarse bucket partitioning usually only results in a lower bound for the instance's makespan. However, by repeatedly refining the bucket partitioning, i.e., splitting a set of buckets into smaller buckets, and resolving the model, we will eventually reach an optimal solution.

We speed this procedure up by employing two primal heuristics at each iteration of the algorithm. The purpose of these heuristics is to construct a valid SI-PTPSP solution from a relaxed solution S yielded by TBR. The first heuristic is the gap closing heuristic (GCH). The goal of GCH is to derive a valid SI-PTPSP solution S' from S s.t. the makespan of S' does not exceed the makespan of S. This way it is guaranteed that S is optimal. However, GCH may not always succeed in deriving such a solution, especially in early iterations of the algorithm.

In case GCH cannot provide a solution, we apply a follow-up heuristic, whose goal is to derive any feasible SI-PTPSP solution from S. If this heuristic cannot close the gap to the TBR bound, we proceed with the bucket refinement process and solve TBR again (see Algorithm 5.1).

Effective refinement strategies are of utter importance for the performance of ITBRA. If the bucket partitioning is too coarse, the primal heuristics may fail to construct a

Algorithm 5.1: 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
```

good bound which increases the number of iterations of ITBRA. On the other hand, if the bucket partitioning is too fine-grained, the number of variables in the TBR model increases too fast, resulting in higher computation times for solving the TBR model. We will see that the quality of a bucket partitioning depends not only on the number of created buckets but also on the structure of the partitioning. Some parts of the time horizon are more important than others and need to be refined to a higher degree. In Section 5.3 we suggest several refinement strategies. Most of these strategies exploit information obtained from the TBR solution and the applied primal heuristics.

The pseudocode of ITBRA is shown in Algorithm 5.1. The individual components of ITBRA will be explained in detail in the next sections.

5.1 Initial Bucket Partitioning

We create the initial bucket partitioning B in such a way that buckets start/end at any time where a resource availability interval starts or ends and at any release time and deadline of the activities. For details see Algorithm 5.2.

5.2 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

Algorithm 5.2: 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:  $\mathcal{T} \leftarrow \mathcal{T} \cup \{W^{\text{start}}_{r,w}, W^{\text{end}}_{r,w} + 1 : r \in R,\ w = 1, \ldots, \omega_r\}$;
4:  $\mathcal{T} \leftarrow \mathcal{T} \cup \{t^{\text{r}}_a, t^{\text{d}}_a : a \in A\}$;
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 $B$;
```

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.

5.2.1 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 $S^{\text{TBR}}_a = \{S^{\text{TBR,min}}_a, \ldots, S^{\text{TBR,max}}_a\}$ of potential starting times, where $S^{\text{TBR,min}}_a = \sum_{c=1}^{\gamma_a} S^{\min}_{a,c} \cdot y^*_{a,c}$ and $S^{\text{TBR,max}}_a = \sum_{c=1}^{\gamma_a} S^{\max}_{a,c} \cdot y^*_{a,c}$. These intervals can in general be further reduced by removing for each $a \in A$ all time slots $t \in S^{\text{TBR}}_a$ violating at least one of the following conditions in relation to the precedence constraints and the calculation of the makespan:

$$\exists t' \in S^{\text{TBR}}_{a'} (t + p_a + L^{\min}_{a,a'} \le t' \le t + p_a + L^{\max}_{a,a'}) \qquad \forall (a, a') \in P \tag{5.1}$$

$$\exists t' \in S^{\text{TBR}}_{a'} (t' + p_{a'} + L^{\min}_{a',a} \le t \le t' + p_{a'} + L^{\max}_{a',a}) \qquad \forall (a', a) \in P \tag{5.2}$$

$$t + p_a \le MS^* \tag{5.3}$$

We repeatedly prune the set of intervals of potential activity starting times for all activities $S^{\text{TBR}} = \{S^{\text{TBR}}_a : a \in A\}$ w.r.t. Conditions (5.1)–(5.3) until no more starting times can be discarded. This is done by Algorithm 5.3 which is based on the well-known AC3 algorithm, see Mackworth [1977]. Algorithm 5.3 utilizes the fact that the precedence graph has to be acyclic. The order in which the activities are considered for pruning, may have a negative impact on the number of iterations of Algorithm 5.3. Therefore, Algorithm 5.3 reverses the pruning order after each iteration.

Algorithm 5.3: PruneStartingTimes

```
   Input : intervals of potential starting times $S^{\text{TBR}} = \{S_a^{\text{TBR}} : a \in A\}$,
           an ordered set of activities $A'$
   Output: $S^{\text{TBR}}$ pruned w.r.t. Conditions (5.1)–(5.3)
1: while $A' \neq \emptyset$ do
2:     $(S^{\text{TBR}}, A') \leftarrow$PruneOrdered$(S^{\text{TBR}}, A')$;
3:     reverse the order of $A'$ ;
4: end while
5: return $S^{\text{TBR}}$;
```

While Algorithm 5.3 determines the order in which the starting times are pruned, Algorithm 5.4 does the actual pruning. For each S_a^{TBR} Algorithm 5.4 removes all times violating Conditions (5.1)–(5.3). The algorithm returns the pruned starting times and a set of activities whose starting times can possibly be pruned again. This set contains the activities a' and a'' s.t. $(a, a'), (a'', a) \in P$ for each S_a^{TBR} that has been pruned.

Activities for which only a single feasible starting time remains can be handled more efficiently since no further pruning is applicable for them. This is particularly relevant during GCH since the pruning is repeated whenever an activity's starting time is fixed. Hence, Algorithm 5.5 partitions the set of all activities into a set A_1 containing all activities that topologically appear before a and a set A_2 containing all activities that topologically appear after a. Then, Algorithm 5.3 is applied on A_1 and A_2.

Note that the pruning algorithms may yield empty intervals for some activities (i.e., intervals with $S_a^{\text{TBR,min}} > S_a^{\text{TBR,max}}$), indicating that there remains no feasible starting time assignment respecting all constraints. In this case the pruning algorithm 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 5.6. 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 (5.1)–(5.3). Remaining unscheduled activities are partitioned into two sets, depending on the reason for which they could not be scheduled. This information is relevant for the bucket refinement process (see Section 5.3).

Algorithm 5.4: PruneOrdered

```
Input : intervals of potential starting times $S^{\mathrm{TBR}} = \{S_a^{\mathrm{TBR}} : a \in A\}$ with
        $S_a^{\mathrm{TBR}} = \{S_a^{\mathrm{TBR,min}}, \dots, S_a^{\mathrm{TBR,max}}\}$,
        an ordered set of activities $A'$ whose corresponding starting time intervals
        have to be be pruned
Output: partially pruned set $S^{\mathrm{TBR}}$ w.r.t. Constraints (5.1)–(5.3), a set of
        activities whose starting times can possibly be pruned again
1:  $A'' \leftarrow \emptyset$; // set of activities whose starting times can
    possibly be pruned again
2:  forall $a \in A'$ do
3:      forall $a' : ((a,a') \in P \;\vee (a',a) \in P) \wedge S_{a'}^{\mathrm{TBR,min}} \leq S_{a'}^{\mathrm{TBR,max}}$ do
4:          $S_a^{\mathrm{TBR,min}} \leftarrow \max\{S_a^{\mathrm{TBR,min}}, S_{a'}^{\mathrm{TBR,min}} + p_{a'} + L_{a',a}^{\min}\}$ ;
5:          $S_a^{\mathrm{TBR,max}} \leftarrow \min\{S_a^{\mathrm{TBR,max}}, S_{a'}^{\mathrm{TBR,max}} + p_{a'} + L_{a',a}^{\max}\}$;
6:          $S_a^{\mathrm{TBR,min}} \leftarrow \max\{S_a^{\mathrm{TBR,min}}, S_{a'}^{\mathrm{TBR,min}} - p_a - L_{a,a'}^{\max}\}$;
7:          $S_a^{\mathrm{TBR,max}} \leftarrow \min\{S_a^{\mathrm{TBR,max}}, S_{a'}^{\mathrm{TBR,max}} - p_a - L_{a',a}^{\min}\}$;
8:          $S_a^{\mathrm{TBR,max}} \leftarrow \min\{S^{\mathrm{TBR,max}}, MS - pa\}$;
9:      end
10:     if new values assigned to $S_a^{\mathrm{TBR,min}}$ or $S_a^{\mathrm{TBR,max}}$ then
11:         $A'' \leftarrow A'' \cup \{a' : (a,a') \in P \;\vee (a',a) \in P\}$;
12:     end if
13: end
14: return $S^{\mathrm{TBR}}, A''$;
```

5.2.2 Activity Block Construction Heuristic (ABCH)

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. The idea of ABCH is to simplify an SI-PTPSP instance by combining the activities of the weakly connected components of the precedence graph into so-called *activity blocks*, i.e., all the activities belonging to one such activity block are statically linked considering the precedence constraints and minimum time lags in between them.

ABCH then tries to construct a schedule using the activity blocks 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.

Applying ABCH on an empty schedule may lead to poor results. Hence, we apply the algorithm on partial schedules S generated by GCH. For this purpose we remove all weakly connected components from S which are not completely scheduled. Details are provided in Algorithm 5.7.

Algorithm 5.5: PruneSinglePoint

```
Input : intervals of potential starting times $S^{\mathrm{TBR}} = \{S_a^{\mathrm{TBR}} : a \in A\}$,
        the set of activities $A'$ in topological order w.r.t. $P$,
        an activity $a$ with $S_a^{\mathrm{TBR,min}} = S_a^{\mathrm{TBR,max}}$
Output: $S^{\mathrm{TBR}}$ pruned w.r.t. violations of conditions (5.1)–(5.3)
1: $A_1 \leftarrow \{a' : a' \in A \wedge a'$topologically appears in $A'$ before $a\} \cup \{a\}$ ;
2: $A_2 \leftarrow \{a' : a' \in A \wedge a'$topologically appears in $A'$ after $a\} \cup \{a\}$ ;
3: sort $A_1$ and $A_2$ topologically ;
4: reverse the order of $A_1$ ;
5: $S^{\mathrm{TBR}} \leftarrow$ PruneStartingTimes($S^{\mathrm{TBR}}, A_1$);
6: $S^{\mathrm{TBR}} \leftarrow$ PruneStartingTimes($S^{\mathrm{TBR}}, A_2$);
7: return $S^{\mathrm{TBR}}$;
```

5.2.3 Greedy Randomized Adaptive Search Procedure (GRASP)

We can improve ABCH by extending it to a GRASP. The approach provides a reasonable balance between being still relatively simple but providing considerably better results than ABCH.

In the following, we will first discuss the construction heuristic of the GRASP and afterwards its local search component. The construction heuristic of the GRASP is a combination of GCH and ABCH. First, we construct a partial schedule using GCH. Then, the schedule is completed with ABCH. The GRASP requires a randomized construction heuristic. Both, GCH and ABCH, can be randomized. The randomization procedure is the similar for both algorithms. Randomization is done by allowing the order in which the activities or activity blocks are scheduled 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. However, note that we also try to compute a primal solution at the very beginning before solving TBR for the first time. Hence, there is no GCH solution available at this point. In this case we randomize ABCH instead.

As mentioned before, many of our bucket refinement strategies exploit information obtained from partial GCH schedules. However, as GCH is randomized in the GRASP, we have multiple schedules to choose from. 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 during GRASP. 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

Algorithm 5.6: Gap closing heuristic (GCH)

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

replaces Lines 5–8 in Algorithm 5.1.

The applied local search component considers a classical 2-exchange neighbourhood on the order of the activity blocks scheduled by ABCH. A best improvement strategy is applied, and 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 Chapter 7. 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

Algorithm 5.7: Activity block construction heuristic (ABCH)

```
    Input: a partial schedule $S^{\text{GCH}}$ computed by GCH
    Output: a feasible schedule $S$ or no solution if $S^{\text{GCH}}$ cannot be completed
 1: $C \leftarrow$ set of subsets of $A$ corresponding to the weakly connected components in the
      precedence graph which are not completely scheduled in $S^{\text{GCH}}$;
 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. 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 $\min_{a \in c} t^{\text{r}}_a$;
 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 $S$;
```

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

5.3 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 only consists of 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 3).

More formally, a refinement of some bucket $B_b \in B$ is given by an ordered set of splitting points $\tau^b = \{\tau^b_1, \ldots, \tau^b_m\} \subseteq \{B^{\text{start}}_b + 1, \ldots, B^{\text{end}}_b\}$ with $\tau^b_1 < \ldots < \tau^b_m$. Based on τ^b we get $|\tau^b| + 1$ new buckets replacing the original one: $\{B^{\text{start}}_b, \ldots, \tau^b_1 - 1\}, \{\tau^b_1, \ldots, \tau^b_2 - 1\}, \ldots, \{\tau^b_m, \ldots, B^{\text{end}}_b\}$. For an example see Figure 5.1.

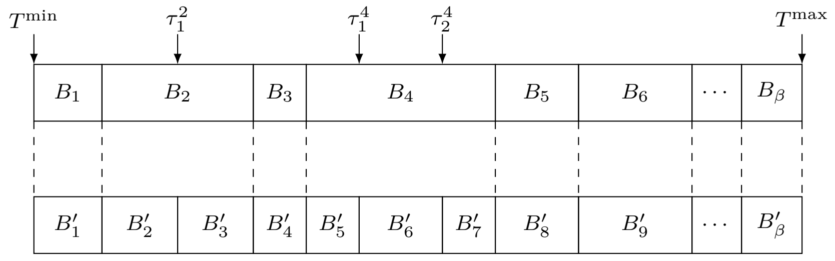


Figure 5.1: An example of a bucket refinement for $\tau^2 = \{\tau_1^2\}$, $\tau^4 = \{\tau_1^4, \tau_2^4\}$, and $\tau^b = \emptyset$ for $b \in I(B) \setminus \{2, 4\}$.

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. A bucket refinement should be done in such a way that the current optimal TBR solution becomes invalid. In this way, it is ensured that in each iteration we obtain a more refined solution. Furthermore, bucket refinements should resolve conflicts between activities that cannot be scheduled together. Therefore, information from constraints that are responsible for such conflicting activities 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.

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 model anymore and therefore cannot be used again. Furthermore, some of the 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 7.2.2.

All Selected (ASEL) We refine all first and last buckets of all bucket sequences selected in the identified 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 of the buckets from ASEL.

All In GCH Schedule (AIGS) Considering the partial schedule S generated by GCH we only refine all first and last buckets of those bucket sequences $C_{a,c}$ whose corresponding activities a are feasibly scheduled in S. The idea is to improve accuracy for the activities that could be scheduled in order to reveal sources of infeasibility w.r.t. the activities that could not be scheduled once TBR is solved the next time.

Incompletely Scheduled Connected Components (ISCC) We split the first and last buckets of all bucket sequences selected in the optimal TBR solution for activities belonging to weakly connected components for which not all activities could be feasibly scheduled by GCH. This approach is conceptually similar to AIGS, however, we additionally take the activity blocks of the unscheduled activities into account to obtain additional candidates for refinement.

Violated Due (VDUE) If GCH fails to schedule all activities, it provides a set of activities A_P that cannot be scheduled due to precedence constraint violations and a set of activities A_R that cannot be scheduled due to resource constraint violations. The basic idea is to refine first and last buckets of bucket sequences selected for activities in the schedule that immediately prevent the activities in A_P and A_R from being scheduled. To identify those activities we again consider the partial schedule S generated by GCH. Let $A^{\text{GCH}} = A \setminus (A_R \cup A_P)$ be the set of feasibly scheduled activities.

Refinements based on resource infeasibilities are derived from sets $N_R(a) = \{a' \in A^{\text{GCH}} : Q_a \cap Q_{a'} \neq \emptyset \wedge \{S_{a'}, \ldots, S_{a'} + p_{a'} - 1\} \cap \{S_a^{\text{TBR,min}}, \ldots, S_a^{\text{TBR,max}} + p_a - 1\} \neq \emptyset\}$ 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 $N_P^-(a) = \{a' : (a', a) \in P \wedge a' \in A^{\text{GCH}}\}$ and $N_P^+(a) = \{a' : (a, a') \in P \wedge a' \in A^{\text{GCH}}\}$ for all $a \in A_P$. Then, calculate:

$$\begin{aligned} N_P(a) = & \arg\max_{a' \in N_P^-(a)} \{S_{a'} + p_{a'} + L_{a',a}^{\min}\} && \cup \quad \arg\min_{a' \in N_P^-(a)} \{S_{a'} + p_{a'} + L_{a',a}^{\max}\} \quad \cup \\ & \arg\min_{a' \in N_P^+(a)} \{S_{a'} - L_{a,a'}^{\max}\} && \cup \quad \arg\max_{a' \in N_P^+(a)} \{S_{a'} - L_{a,a'}^{\min}\} \end{aligned} \tag{5.4}$$

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.

Last Used Bucket (LUSED) Refining the first and last bucket(s) used in selected bucket sequences that are directly responsible for the objective value appears promising for improving the lower bound in the next iteration. With the makespan as objective, these are all bucket sequences that fulfill equations (4.54) or equations (4.82) at equality.

Critical Path (CPATH) We consider a complete directed graph with all activities as nodes. A *critical path* in this graph is a sequence of activities whose selection of bucket sequences successively depends on each other and who are jointly directly responsible for the obtained objective value.

Such a critical path is obtained in reverse order as follows. We start with an activity whose selected bucket sequence fulfills equation (4.54) without slack (compare LUSED). If path inequalities are used, we start with an activity whose selected bucket sequence fulfills equation (4.82) without slack.

In general, an activity $a \in A$ on a critical path may have a predecessor if and only if its selected bucket sequence $C_{a,c}$ is not the first one possible w.r.t. C_a, i.e., $C_{a,1}$, when considering no other activities. In this case, we try to determine a predecessor as follows.

First we deal with activities reachable via the precedence graph. We only consider initial resource availabilities for the current activity and do not take the consumption of the other activities into account. Feasible predecessors w.r.t. a are activities a' with $(a', a) \in P$ s.t. a' is assigned to a bucket sequence preventing selection of an earlier sequence for a (considering the ordering of C_a). Note that multiple candidates might exist; we simply take the first one identified.

Every time no further predecessors can be identified using the precedence graph we resort to information obtained from the resource consumptions instead. We consider bucket sequence $C_{a,c-1}$ directly preceding the one selected by TBR w.r.t. C_a. If there is a resource $r \in Q_a$ required by other activities so that less than $z^{\min}_{a,b,c-1}$ capacity would remain in a bucket $b \in C_{a,c-1}$ for activity a, then these respective other activities are feasible predecessors of a. Among these candidates we choose one for which the bucket sequence c' selected in TBR's solution maximizes $\mathrm{bfirst}(a, c-1) - \mathrm{bfirst}(a', c')$.

Finally, we split first and last buckets of the bucket sequences selected by TBR for all activities being part of the determined critical path.

All Critical Paths (ACPATHS) As already mentioned above, there is in general no unique critical path. Therefore, it appears to be also meaningful to consider all activities involved in any critical path. To derive them, we start from all activities whose selected bucket sequence fulfills equation (4.54) (or equation (4.82)) without slack, and determine all predecessors in a recursive manner following the same considerations as in CPATHS.

Combinations Note that approaches LUSED, CPATH and ACPATHS might not always be able to identify feasible, i.e., non-unit, buckets for refinement. Thus, we need to combine them with some of the other strategies to avoid getting stuck. Additionally, it also makes sense in general to consider combinations of the above strategies to obtain a larger variety of refinement candidates. To reduce the already large number of possible options we only consider combinations for the strategies that cannot be applied on their own in our experiments.

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 bfirst(a, c) and blast(a, c), we consider the splitting positions $\lceil (S^{\min}_{a,c} + S^{\max}_{a,c})/2 \rceil$ and $\lceil (S^{\min}_{a,c} + S^{\max}_{a,c})/2 \rceil + p_a$, 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 since the resulting buckets might still be too large to reveal certain sources of infeasibility.

Latest Start/Earliest End (LSEE) Instead of doing the splitting so that an activity's interval of potential starting times is bisected, we split in this variant at times $S^{\max}_{a,c}$ and $S^{\min}_{a,c} + p_a$. Note that the obtained splitting positions are typically far from the center of the bucket and often even lead to unit buckets. However, this strategy is still worth considering since the border cases are explicitly split off.

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 bfirst(a, c) at the activity's starting time S_a and 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.

As 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 or LSEE if SET is not applicable. The obtained strategies are denoted by SET+B and SET+LSEE, respectively.

Selecting Splitting Positions

The strategies introduced above may yield several splitting positions for a single bucket, especially since one and 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., $\tau' = \arg\min_{t \in \tau^b} \left| \frac{B_b^{\text{start}} + B_b^{\text{end}}}{2} - t \right|$; ties are broken according to the order 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.

Median Partition Refinement (MPR) We first partition τ^b into two sets at $\bar{t} = \frac{B_b^{\text{start}} + B_b^{\text{end}}}{2}$: Let $\tau^{b,\text{l}} = \{t \in \tau^b : t \leq \bar{t}\}$ and $\tau^{b,\text{r}} = \{t \in \tau^b : t > \bar{t}\}$. The refinement we apply splits the bucket into three buckets using the median of each set, i.e., $\{\text{median}(\tau^{b,\text{l}}), \text{median}(\tau^{b,\text{r}})\}$. Note that either of the sets might be empty and then we only obtain two sub-buckets.

The idea of partitioning the potential splitting positions in this way is to give candidate positions close to either boundary of the bucket equal chances of being selected. This is motivated by the fact that 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. Both cases might be equally relevant and thus it makes sense to apply a bucket refinement considering each of them.

Centered Partition Refinement (CPR) We again partition τ^b into two sets at $\bar{t} = \frac{B_b^{\text{start}} + B_b^{\text{end}}}{2}$. Let $\tau^{b,\text{l}} = \{t \in \tau^b : t \leq \bar{t}\}$ and $\tau^{b,\text{r}} = \{t \in \tau^b : t > \bar{t}\}$. 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 only apply a single split.

The idea is similar to the one of MPR; however, this time we prefer splitting positions close to the center of the bucket. This might be beneficial in situations where candidate positions are (too) strongly clustered at the boundaries of the bucket.

Figure 5.2 provides an overview of the discussed bucket selection, splitting position identification, and splitting position selection strategies and their possible combinations.

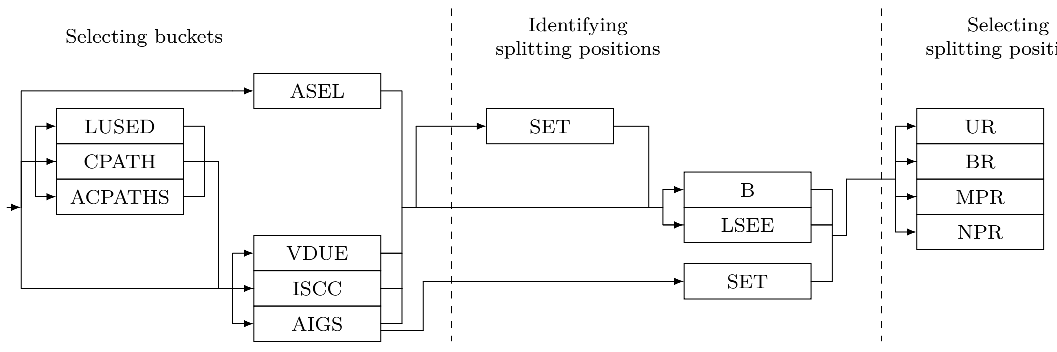


Figure 5.2: Overview of the proposed strategies to perform a bucket refinement and how they can be combined.

CHAPTER 6

Implementation Details

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

6.1 Preprocessing Activity Starting Times

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

$$T_a = \{t \in T : t_a^{\mathrm{r}} \leq t \leq t_a^{\mathrm{d}} - p_a, \forall r \in Q_a, t' \in Y_a(t)(t' \in W_r)\}$$

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:

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

We can use constraint propagation for this purpose similar as in GCH to achieve arc consistency w.r.t. these conditions. 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

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

6.2 On Determining Big-M Constants for DEF

Choosing smallest possible Big-M constants for Inequalities (4.10)–(4.13) in DEF is important for making its LP relaxation as tight as possible. To this end we compute bounds on the number of events that precede and succeed each activity event. Let k_a^{S} be the event at which activity $a \in A'$ starts and k_a^{F} be the event at which it ends. Now let K_k^{pre} and K_k^{suc} be sets of events that must precede and succeed event k in any feasible solution.

Note that it is not known in advance which event corresponds to which activity. Consequently, it is difficult to determine complete sets K_k^{pre} and K_k^{suc}. However, using release times, deadlines, and precedence relations we can derive reasonable, but in general incomplete sets K_k^{pre} and K_k^{suc} efficiently. Since the events are ordered, it follows that the ith event $k_i \in K$ is contained in K_k^{pre} for $i = 1, \dots, |K_k^{\mathrm{pre}}|$ and k_i is contained in K_k^{suc} for $i = |K| - |K_k^{\mathrm{suc}}| + 1, \dots, |K|$. Thus, we can set the big-M constants in Inequalities (4.10)–(4.13) as follows:

$$M_{a,k}^{(4.10)} = \begin{cases} 0 & \text{for } k \in \{k_1, \dots, k_{|K^{\mathrm{pre}}_{k_a^{\mathrm{S}}}|}\} \\ T^{\max} - t_a^r & \text{otherwise} \end{cases} \tag{6.1}$$

$$M_{a,k}^{(4.11)} = \begin{cases} 0 & \text{for } k \in \{k_{|K|-|K^{\mathrm{suc}}_{k_a^{\mathrm{F}}}|+1}, \dots, k_{|K|}\} \\ t_a^d - p_a - T^{\min} & \text{otherwise} \end{cases} \tag{6.2}$$

$$M_{a,k}^{(4.12)} = \begin{cases} 0 & \text{for } k \in \{k_1, \dots, k_{|K^{\mathrm{pre}}_{k_a^{\mathrm{F}}}|}\} \\ T^{\max} - t_a^r - p_a & \text{otherwise} \end{cases} \tag{6.3}$$

$$M_{a,k}^{(4.13)} = \begin{cases} 0 & \text{for } k \in \{k_{|K|-|K^{\mathrm{suc}}_{k_a^{\mathrm{F}}}|+1}, \dots, k_{|K|}\} \\ t_a^d - T^{\min} & \text{otherwise} \end{cases} \tag{6.4}$$

Obtaining complete sets K_k^{pre} and K_k^{suc} is usually difficult. Thus, we determine reasonable, but in general incomplete sets efficiently by Algorithm 6.1. Over all $a' \in A'$ together, the procedure requires time $O(|A'|^2)$.

The estimation for K_k^{suc} is analogous to the estimation of K_k^{pre}.

6.3 Computing Bucket Sequences

Algorithm 6.2 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 $S_{a,c}^{\min}$. 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,

Algorithm 6.1: Efficiently determining not necessarily complete sets $K^{\text{pre}}_{k^{\text{S}}_a}$ of events that must precede a given activity start event k^{S}_a in DEF.

```
    Input: activity $a \in A'$
    Output: sets $K^{\text{pre}}_{k^{\text{S}}_a}$ and $K^{\text{pre}}_{k^{\text{F}}_a}$
1:  $K^{\text{pre}}_{k^{\text{S}}_a}, K^{\text{pre}}_{k^{\text{F}}_a} \leftarrow \emptyset$ ;
2:  forall $a' \in A' \setminus \{a\}$ do
3:      if $t^d_{a'} - p_{a'} < t^r_a \;\vee\; (t^d_{a'} - p_{a'} = t^r_a \wedge a' < a)$ then
4:          $K^{\text{pre}}_{k^{\text{S}}_a} \leftarrow K^{\text{pre}}_{k^{\text{S}}_a} \cup \{k^{\text{S}}_{a'}\}$;
5:          $K^{\text{pre}}_{k^{\text{F}}_a} \leftarrow K^{\text{pre}}_{k^{\text{F}}_a} \cup \{k^{\text{S}}_{a'}\}$;
6:      end if
7:      if $t^d_{a'} < t^r_a \;\vee\; t^d_{a'} = t^r_a \wedge a' < a$ then
8:          $K^{\text{pre}}_{k^{\text{S}}_a} \leftarrow K^{\text{pre}}_{k^{\text{S}}_a} \cup \{k^{\text{F}}_{a'}\}$;
9:          $K^{\text{pre}}_{k^{\text{F}}_a} \leftarrow K^{\text{pre}}_{k^{\text{F}}_a} \cup \{k^{\text{F}}_{a'}\}$;
10:     end if
11:     if $t^d_{a'} - p_{a'} < t^r_a + p_a \;\vee\; t^d_{a'} - p_{a'} = t^r_a + p_a \wedge a' < a$ then
12:         $K^{\text{pre}}_{k^{\text{F}}_a} \leftarrow K^{\text{pre}}_{k^{\text{F}}_a} \cup \{k^{\text{S}}_{a'}\}$;
13:     end if
14:     if $t^d_{a'} < t^r_a + p_a \;\vee\; t^d_{a'} = t^r_a + p_a \wedge a' < a$ then
15:         $K^{\text{pre}}_{k^{\text{F}}_a} \leftarrow K^{\text{pre}}_{k^{\text{F}}_a} \cup \{k^{\text{F}}_{a'}\}$;
16:     end if
17: end
18: return $K^{\text{pre}}_{k^{\text{S}}_a}, K^{\text{pre}}_{k^{\text{F}}_a}$;
```

we try to find the earliest time point that guarantees that we start in b^{first} and finish in the earliest bucket succeeding b^{last}. If no such time point exists, we proceed with the earliest time slot in bucket $b^{\text{first}} + 1$ instead. 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 $T^{\text{s}}_{a,c} = \{S^{\min}_{a,c}, \ldots, S^{\max}_{a,c}\} \cap T_a$. We can discard all bucket sequences for which $T^{\text{s}}_{a,c} = \emptyset$. Moreover, $S^{\min}_{a,c}$ and $S^{\max}_{a,c}$ can be strengthened as follows:

$$S^{\min}_{a,c} = \min(T^{\text{s}}_{a,c})$$
$$S^{\max}_{a,c} = \max(T^{\text{s}}_{a,c})$$

Algorithm 6.2: 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:      $b^{\mathrm{first}} \leftarrow b : t \in B_b$;
6:      $b^{\mathrm{last}} \leftarrow b : t + p_a - 1 \in B_b$;
7:      $C_{a,c} \leftarrow \{B_{b^{\mathrm{first}}}, \dots, B_{b^{\mathrm{last}}}\}$;
8:      $S^{\min}_{a,c} \leftarrow t$;
9:      if $b^{\mathrm{first}} = b^{\mathrm{last}}$ then
10:         $z^{\min}_{a,b^{\mathrm{last}},c} \leftarrow p_a$;
11:         $z^{\max}_{a,b^{\mathrm{last}},c} \leftarrow p_a$;
12:         $t \leftarrow B^{\mathrm{end}}_{b^{\mathrm{last}}} - p_a + 2$;
13:     else
14:         $z^{\max}_{a,b^{\mathrm{first}},c} \leftarrow B^{\mathrm{end}}_{b^{\mathrm{first}}} - t + 1$;
15:         $z^{\min}_{a,b^{\mathrm{last}},c} \leftarrow S^{\min}_{a,c} + p_a - B^{\mathrm{start}}_{b^{\mathrm{last}}}$;
16:         $\delta \leftarrow \min\left\{z^{\max}_{a,b^{\mathrm{first}},c} - 1, \min\left\{B^{\mathrm{end}}_{b^{\mathrm{last}}}, t^{\mathrm{d}}_a - 1\right\} - \left(S^{\min}_{a,c} + p_a - 1\right)\right\}$;
17:         $z^{\min}_{a,b^{\mathrm{first}},c} \leftarrow z^{\max}_{a,b^{\mathrm{first}},c} - \delta$;
18:         $z^{\max}_{a,b^{\mathrm{last}},c} \leftarrow z^{\min}_{a,b^{\mathrm{last}},c} + \delta$;
19:         $t \leftarrow S^{\min}_{a,c} + \delta + 1$;
20:     end if
21:     $S^{\max}_{a,c} = B^{\mathrm{end}}_{b^{\mathrm{first}}} - z^{\min}_{a,b^{\mathrm{first}},c} + 1$;
22:     $C_a \leftarrow C_a \cup \{C_{a,c}\}$;
23:     $c \leftarrow c + 1$;
24: end while
25: return $C_a$;
```

6.4 Valid Inequalities

As already mentioned we only consider the simplified version of the clique inequalities (4.78) and (4.79) 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 (4.80)–(4.82) is in general exponential. In favour of keeping the model compact we avoided dynamic separation and only consider 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 vertices 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:

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

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

We then add Inequalities (4.80) and (4.82) only for paths $\pi_{a,a'} \in \Pi_{L^{\min}}$ and Inequalities (4.81) for paths $\pi_{a,a'} \in \Pi_{L^{\max}}$.

CHAPTER 7

Computational Results

In this chapter we are going to present the computational results for the considered algorithms with their variants. We first show how our test instances are generated. Then, we provide details on the actually used configurations. Finally, we present the obtained results.

7.1 Test Instances

The benchmark instances are motivated by the real world patient scheduling scenario at cancer treatment center MedAustron that requires planning 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, irradiation rooms, 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 that is required by exactly one activity of each treatment. This 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. 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 $\alpha \in \{20, 30, \ldots, 100\}$. 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 needed 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 $\{100, \ldots, 10000\}$. 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 $\{0, \ldots, \sum_{a \in A}(p_a + 10000)\}$, by randomly selecting a starting time for the first activity of each connected component. For the subsequent activities a random time lag in $\{L^{\min}_{a,a'}, \ldots, L^{\max}_{a,a'}\}$ is enforced. If a determined starting time of an activity conflicts with an already scheduled one, then 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. The functions $f^{hard}(\alpha) = \frac{40-0.08\cdot\alpha}{80} \cdot \sum_{a \in A}(p_a + 10000)$ and $f^{easy}(\alpha) = 0.8 \cdot f^{hard}(\alpha)$ control the distance between release time and deadline. For a preliminary starting time S'_a of an activity a and a difficulty $\mathit{diff} \in \{\mathit{easy}, \mathit{hard}\}$, the tentative release time is calculated by $t^{\mathrm{r}}_a = \max(0, S'_a - f^{\mathit{diff}}(\alpha))$. The tentative deadline is calculated by $t^{\mathrm{d}}_a = \min(\sum_{a \in A}(p_a + 10000), S'_a + p_a + f^{\mathit{diff}}(\alpha))$. Up to this point the instance is trivially solvable. To make the instance more challenging, the tentative release times and deadlines are scaled by a factor $s \in (0, 1]$. We have chosen a scaling factor depending on the number of activities $s(\alpha) = \frac{44.6-0.23\cdot\alpha}{80}$.

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 $\{700, \ldots, 1500\}$. The positions of these unavailability windows are chosen uniformly at random from the set $\{0, \ldots, T^{\max}\}$.

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 7.1. 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.

7.2 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

Table 7.1: 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 per instance set.

set	$T^{\max}$	ρ	chains
e_{20}	104 649	7	4
e_{30}	138 808	8	6
e_{40}	169 642	9	8
e_{50}	198 386	10	10
e_{60}	220 792	11	12
e_{70}	244 279	12	14
e_{80}	271 461	13	16
e_{90}	293 110	14	18
e_{100}	316 316	15	20

set	$T^{\max}$	ρ	chains
h_{20}	104 575	7	4
h_{30}	137 745	8	6
h_{40}	167 003	9	8
h_{50}	201 269	10	10
h_{60}	220 606	11	12
h_{70}	244 788	12	14
h_{80}	271 327	13	16
h_{90}	289 278	14	18
h_{100}	317 324	15	20

using Gurobi 7 with a single thread.

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.

7.2.1 Parametrization of the Primal Heuristics

GRASP from Section 5.2.3 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 $k^{\mathrm{grand}}_{\mathrm{ABCH}}$ has been set to 8 for all benchmark instances. We applied irace (López-Ibáñez et al. [2016]) to determine this value. However, it turned out that the performance of our GRASP is very robust against changes to $k^{\mathrm{grand}}_{\mathrm{ABCH}}$.

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 $k_{\text{GCH}}^{\text{grand}}$ and $k_{\text{ABCH}}^{\text{grand}}$ of the embedded GRASP have been determined experimentally starting with the values from the stand-alone variant. For the parameter $k_{\text{GCH}}^{\text{grand}}$ we first assumed a value of $k_{\text{GCH}}^{\text{grand}} = 5 \cdot k_{\text{ABCH}}^{\text{grand}}$ as all activity chains in the test instances consist of five activities. Afterwards, we fine-tuned these parameters by iterative adjustment. The parameter $k_{\text{ABCH}}^{\text{grand}}$ is set to 6 and $k_{\text{GCH}}^{\text{grand}}$ 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 than 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.

7.2.2 Comparison of Bucket Refinement Strategies

Due to the large number of possible combinations of refinement techniques (cf. Figure 5.2) we did not test every variant. Instead we employ a local search strategy to identify good options: starting from an initial reference strategy, we investigate the impact of replacing each of the three parts of the refinement process. At each stage, we choose the strategy with the best performance as new reference strategy.

As initial refinement strategy we choose ASEL,B,UR as it is one of the most straightforward refinement strategies as it refines all bucket sequences in the identified optimal TBR solution. Moreover, refinement decisions are made by only considering the TBR solution.

We evaluate the performance of a refinement strategy based on the size of the optimality gaps, the number of solved instances, and the computation times.

Starting with the strategy ASEL,B,UR, we first replace the strategy deciding which buckets to refine. To this end, we replace ASEL with ISCC, AIGS, and VDUE. The results of the corresponding strategies are shown in Table 7.2. Note that we evaluate the bucket selection strategies that cannot be applied alone at a later point (see Tables 7.6–7.7).

VDUE,B,UR dominates the other strategies in every aspect, which indicates that the strategy deciding which buckets to refine has a large impact on the performance of the algorithm. Another interesting result is that ASEL,B,UR and ISCC,B,UR produce the

Table 7.2: Comparison of ASEL, ISCC, AIGS, and VDUE in combination with B and UR. We consider the average optimality gaps (gap), the number of solved instances (opt) and the median computation times in seconds (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. The best values per instance set are highlighted bold.

	ASEL			ISCC			AIGS			VDUE		
	B			B			B			B		
	UR			UR			UR			UR		
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**	**12**	**0.0**	**15**	22	**0.0**	**15**	**12**
e_{30}	0.6	14	219	0.6	14	206	2.9	13	197	**0.0**	**15**	**129**
e_{40}	4.6	**10**	836	4.6	**10**	860	4.6	9	243	**4.2**	**10**	**92**
e_{50}	1.1	14	189	1.1	14	**188**	0.5	14	758	**0.0**	**15**	**188**
e_{60}	**1.2**	**14**	82	**1.2**	**14**	83	2.1	12	171	**1.2**	**14**	**63**
e_{70}	2.1	8	6957	2.1	8	6965	2.0	8	3385	**1.4**	**11**	**614**
e_{80}	2.0	7	tl	2.0	7	tl	1.4	9	2888	**0.6**	**11**	**1906**
e_{90}	1.7	6	tl	1.7	6	tl	1.7	6	tl	**1.6**	**7**	tl
e_{100}	**0.9**	5	tl	**0.9**	5	tl	1.4	5	tl	1.3	**6**	tl
summary	1.6	93	836	1.6	93	860	1.8	91	758	**1.1**	**104**	**188**
h_{20}	**0.0**	**15**	17	**0.0**	**15**	16	**0.0**	**15**	**13**	**0.0**	**15**	18
h_{30}	10.7	9	4341	10.7	9	4225	10.5	7	tl	**6.4**	**12**	**1045**
h_{40}	13.8	4	tl	13.8	4	tl	11.6	4	tl	**7.0**	**6**	tl
h_{50}	**18.4**	**2**	tl	**18.4**	**2**	tl	19.4	**2**	tl	19.0	**2**	tl
h_{60}	16.8	1	tl	16.8	1	tl	16.4	**2**	tl	**15.0**	**2**	tl
h_{70}	21.1	0	tl	21.1	0	tl	21.0	0	tl	**19.8**	**2**	tl
h_{80}	**14.6**	**1**	tl	**14.6**	**1**	tl	**14.6**	**1**	tl	**14.6**	**1**	tl
h_{90}	10.5	**1**	tl	10.5	**1**	tl	10.4	**1**	tl	**9.6**	**1**	tl
h_{100}	**10.6**	**0**	tl	**10.6**	**0**	tl	**10.6**	**0**	tl	**10.6**	**0**	tl
summary	12.9	33	tl	12.9	33	tl	12.7	32	tl	**11.3**	**41**	tl

same results. The reason for this behaviour is that GCH fails to completely schedule any whole activity block. Hence, ISCC selects the exact same buckets to refine as ASEL. This is however an instance specific property and ASEL,B,UR and ISCC,B,UR may indeed produce different results for other test instances. Note that for the set of easy instances AIGS,B,UR produces worse optimality gaps than ASEL,B,UR. Hence, refining fewer buckets does not always lead to better results. However, considering VDUE, it also becomes evident that refining only a small set of carefully chosen buckets greatly improves the quality of the solutions.

Next, we examine how the results are affected when refining buckets at different positions. We evaluate the strategies for identifying splitting positions B, LSEE, and SET+B, using VDUE,B,UR as reference model. The results are shown in Table 7.3. VDUE,LSEE,UR performs worse than the reference strategy. LSEE refines more buckets than B but does not incorporate as much information as SET+B. Consequently, LSEE does not seem to be a good strategy for identifying splitting positions. Moreover, LSEE and SET+LSEE seem to be inferior to B and SET+B, correspondingly, implying that the TBR model

Table 7.3: Comparison of B, LSEE, SET+B, and SET+LSEE in combination with VDUE and UR. We consider the average optimality gaps (gap), the number of solved instances (opt) and the median computation times in seconds (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. The best values per instance set are highlighted bold.

	VDUE			VDUE			VDUE			VDUE		
	B			LSEE			SET+B			SET+LSEE		
	UR			UR			UR			UR		
set	gap[%]	opt	$t[s]$	gap[%]	opt	$t[s]$	gap[%]	opt	$t[s]$	gap[%]	opt	$t[s]$
e_{20}	**0.0**	**15**	12	1.7	12	12	**0.0**	**15**	**11**	**0.0**	**15**	15
e_{30}	**0.0**	**15**	129	2.9	13	77	0.6	14	84	0.6	14	**48**
e_{40}	**4.2**	**10**	92	4.6	9	283	4.6	**10**	**72**	4.4	**10**	247
e_{50}	**0.0**	**15**	188	**0.0**	14	87	**0.0**	**15**	89	**0.0**	**15**	**76**
e_{60}	**1.2**	**14**	63	1.5	13	64	**1.2**	**14**	**56**	**1.2**	**14**	70
e_{70}	1.4	11	614	0.8	11	1601	**0.7**	**12**	**446**	1.4	**12**	990
e_{80}	**0.6**	11	1906	**0.6**	10	487	0.7	**12**	774	0.9	10	**146**
e_{90}	1.6	7	tl	**1.4**	**9**	1766	1.6	**9**	**1103**	1.6	7	tl
e_{100}	1.3	6	tl	1.1	8	3396	1.2	**9**	**833**	**1.0**	**9**	1419
summary	**1.1**	104	188	1.6	99	283	1.2	**110**	**89**	1.2	106	146
h_{20}	**0.0**	**15**	18	**0.0**	**15**	**12**	**0.0**	**15**	17	**0.0**	**15**	15
h_{30}	**6.4**	**12**	1045	11.1	8	5889	6.7	**12**	575	6.6	11	**476**
h_{40}	**7.0**	6	tl	8.1	**7**	tl	10.3	6	tl	8.6	**7**	tl
h_{50}	19.0	2	tl	18.3	2	tl	**18.2**	**4**	tl	18.3	3	tl
h_{60}	**15.0**	2	tl	15.6	**3**	tl	15.6	**3**	tl	15.7	**3**	tl
h_{70}	19.8	**2**	tl	20.7	0	tl	**19.5**	1	tl	22.1	0	tl
h_{80}	14.6	1	tl	14.4	1	tl	**14.2**	1	tl	14.6	**2**	tl
h_{90}	9.6	**1**	tl	**9.5**	**1**	tl	10.3	**1**	tl	**9.5**	**1**	tl
h_{100}	**10.6**	**0**	tl	**10.6**	**0**	tl	**10.6**	**0**	tl	**10.6**	**0**	tl
summary	**11.3**	41	tl	12.0	37	tl	11.7	**43**	tl	11.8	42	tl

profits less from splitting buckets at positions chosen by LSEE and SET+LSEE than positions chosen by B or SET+B.

While VDUE,B,UR produces the better optimality gaps, VDUE,SET+B,UR is able to solve more instances to optimality. The faster computation times of VDUE,SET+B,UR seem to be a direct consequence of the higher number of solved instances. The pure binary approach requires fewer refinements per iteration, consequently the strategy is able to process more iterations than VDUE,SET+B,UR. A larger number of iterations usually results in a higher degree of convergence and thus a smaller optimality gap. However, VDUE,B,UR struggles to completely close the gap. The combination of SET and B tries to address this shortcoming by choosing refinement positions in a more elaborate way which, however, results in a larger number of refinements per iteration. Therefore, VDUE,SET+B,UR is not able to process as many iterations as VDUE,B,UR. Since it is not clear which of these approaches is preferable, we use both strategies as reference model for the remaining evaluation stages.

Next, we evaluate the strategies for selecting splitting positions UR, BR, CPR, and MPR. In Table 7.4 we provide the results with VDUE,B,UR as reference model. The results with VDUE,SET+B,UR as reference model are shown in Table 7.5. The experiments show that UR and BR work better in combination with B, while CPR and MPR provide better results when combined with SET+B. The easy instances seem to generally profit from further reducing the number of refinements per iteration. This, however, is not true for the hard instances as VDUE,B,UR features the best gaps together with VDUE,SET+B,CPR. This shows that especially for the hard instances it is crucial to find a good balance between the number of refinements per iteration and its associated information gain. Refining more buckets may provide more information but also increases the model size of TBR. Hence, VDUE,SET+B,UR performs worse than VDUE,B,UR. By refining fewer buckets per iteration the size of the TBR grows slower. However, refining fewer buckets may result in a TBR model that barely profits from the additional buckets. This becomes especially evident when comparing the performance of VDUE,B,CPR and VDUE,SET+B,CPR on the hard instances as VDUE,SET+B,CPR provides significantly better results.

In total, VDUE,SET+B,MPR yields the best results for the easy instance sets, while VDUE,SET+B,CPR yields the best results for the hard instance sets. It is, however, not clear whether CPR or MPR is the better strategy in general.

Last, we evaluate the bucket selection strategies that cannot be applied alone, i.e., LUSED, CPATH, and ACPATHS. First, we use VDUE,SET+B,CPR as reference model. Afterwards, we evaluate these strategies again, using VDUE,SET+B,MPR as reference model. Results for VDUE,SET+B,CPR are shown in Table 7.6 and the results for VDUE,SET+B,MPR are shown in Table 7.7. The results indicate that adding further selection strategies results in better results for some instance sets. However, in general the additional selection strategies have no positive impact on the performance of ITBRA. The additional selection strategies reduce the total number of iterations in general but also greatly increase the number of buckets which seems to be the reason for their bad performance.

Summing up, we can say that the bucket selection strategy has by far the highest impact on the performance. The results show that the quality of a bucket refinement is much more important than the number of refinements. Neither a strategy with a high number of refinements nor a strategy with a low number of refinements works particularly well in general.

Moreover, it turns out that the strategies, that show a good performance, follow two different approaches for solving an instance. The first approach is to keep the number of refinements as small as possible in order to achieve a high number of iterations. This approach usually produces low optimality gaps but struggles to completely close the gap. The other approach is to try closing the optimality gap as fast as possible, by allowing a larger number of refinements. This usually yields a higher number of solved instances but also results in larger optimality gaps due to the faster growth of the model.

Table 7.4: Comparison of UR, BR, CPR, and MPR in combination with VDUE and B. We consider the average optimality gaps (gap), the number of solved instances (opt) and the median computation times in seconds (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. The best values per instance set are highlighted bold.

	VDUE			VDUE			VDUE			VDUE		
	B			B			B			B		
	UR			BR			CPR			MPR		
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**	19	**0.0**	**15**	27	**0.0**	**15**	14
e_{30}	**0.0**	**15**	129	**0.0**	**15**	135	**0.0**	**15**	**91**	**0.0**	**15**	156
e_{40}	4.2	**10**	**92**	**2.7**	**10**	267	4.5	**10**	200	4.5	**10**	178
e_{50}	**0.0**	**15**	188	1.1	14	187	**0.0**	**15**	183	**0.0**	**15**	**174**
e_{60}	**1.2**	**14**	63	**1.2**	13	**54**	**1.2**	**14**	100	**1.2**	13	75
e_{70}	1.4	11	**614**	0.5	**14**	923	0.4	13	742	**0.0**	**14**	1494
e_{80}	**0.6**	**11**	1906	**0.6**	10	3064	0.8	10	**1197**	0.8	10	2287
e_{90}	**1.6**	7	tl	1.7	7	tl	**1.6**	6	tl	**1.6**	**8**	**6710**
e_{100}	1.3	6	tl	1.1	**7**	tl	**0.7**	**7**	tl	1.2	**7**	tl
summary	1.1	104	188	**1.0**	105	267	**1.0**	105	200	**1.0**	**107**	**178**
h_{20}	**0.0**	**15**	18	**0.0**	**15**	**17**	**0.0**	**15**	22	**0.0**	**15**	22
h_{30}	**6.4**	**12**	**1045**	6.6	11	1088	**6.4**	**12**	1860	**6.4**	**12**	1411
h_{40}	**7.0**	6	tl	9.1	6	tl	8.3	6	tl	8.2	**7**	tl
h_{50}	19.0	2	tl	17.8	**4**	tl	**17.0**	**4**	tl	18.4	3	tl
h_{60}	**15.0**	2	tl	15.4	**3**	tl	16.5	**3**	tl	16.1	2	tl
h_{70}	**19.8**	**2**	tl	20.1	1	tl	20.3	0	tl	20.1	1	tl
h_{80}	**14.6**	**1**	tl	**14.6**	**1**	tl	**14.6**	**1**	tl	**14.6**	**1**	tl
h_{90}	9.6	**1**	tl	9.2	**1**	tl	9.8	**1**	tl	**8.9**	**1**	tl
h_{100}	**10.6**	**0**	tl	**10.6**	**0**	tl	**10.6**	**0**	tl	**10.6**	**0**	tl
summary	**11.3**	41	tl	11.5	**42**	tl	11.5	**42**	tl	11.5	**42**	tl

For our test instances the second approach seems to be slightly superior to the first approach. We further investigate the difference between these two approaches in Table 7.8. We have chosen the refinement strategy VDUE,B,CPR for representing the first approach and VDUE,SET+B,CPR for representing the second approach. Moreover, we also compare both strategies to the inferior strategies ASEL,B,UR and AIGS,B,UR. In particular we consider the increase in the number of buckets, the number of iterations, 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.

Strategy ASEL,B,UR and AIGS,B,UR generate significantly more buckets than the remaining approaches. However, the high number of buckets results from different reasons for these strategies. ASEL,B,UR generates a high number of new buckets in each iteration , which 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

Table 7.5: Comparison of UR, BR, CPR, and MPR in combination with VDUE and SET+B. We consider the average optimality gaps (gap), the number of solved instances (opt) and the median computation times in seconds (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. The best values per instance set are highlighted bold.

	VDUE SET+B UR			VDUE SET+B BR			VDUE SET+B CPR			VDUE SET+B MPR		
set	gap[%]	opt	$t[s]$	gap[%]	opt	$t[s]$	gap[%]	opt	$t[s]$	gap[%]	opt	$t[s]$
e_{20}	**0.0**	**15**	**11**	**0.0**	**15**	19	**0.0**	**15**	13	**0.0**	**15**	14
e_{30}	0.6	14	84	**0.0**	**15**	59	0.6	14	66	**0.0**	**15**	**49**
e_{40}	4.6	10	**72**	4.7	10	172	4.8	10	160	**4.5**	**11**	281
e_{50}	**0.0**	**15**	89	1.1	14	110	**0.0**	**15**	105	**0.0**	**15**	**78**
e_{60}	1.2	**14**	**56**	**1.1**	**14**	64	1.2	**14**	78	1.2	**14**	75
e_{70}	0.7	12	446	0.9	12	**325**	**0.3**	**14**	528	0.6	11	460
e_{80}	0.7	12	774	0.9	11	295	**0.4**	**13**	**266**	0.5	**13**	1081
e_{90}	1.6	9	1103	1.5	9	**715**	1.5	9	1908	**1.3**	**10**	1835
e_{100}	1.2	**9**	**833**	1.2	8	1125	1.2	8	4740	**1.1**	**9**	2845
summary	1.2	110	**89**	1.3	108	172	1.1	112	160	**1.0**	**113**	281
h_{20}	**0.0**	**15**	**17**	**0.0**	**15**	24	**0.0**	**15**	20	**0.0**	**15**	24
h_{30}	6.7	**12**	**575**	7.7	10	856	6.4	**12**	985	**6.3**	**12**	720
h_{40}	10.3	6	tl	9.6	6	tl	**8.1**	**7**	tl	9.1	6	tl
h_{50}	**18.2**	**4**	tl	**18.2**	**4**	tl	**18.2**	**4**	tl	**18.2**	**4**	tl
h_{60}	15.6	3	tl	15.7	2	tl	**15.3**	**4**	tl	15.5	2	tl
h_{70}	**19.5**	1	tl	20.3	1	tl	19.9	**3**	tl	19.8	2	tl
h_{80}	**14.2**	1	tl	14.3	1	tl	14.3	**2**	tl	**14.2**	1	tl
h_{90}	10.3	**1**	tl	**9.3**	**1**	tl	10.0	**1**	tl	10.1	**1**	tl
h_{100}	**10.6**	**0**	tl	**10.6**	**0**	tl	**10.6**	**0**	tl	**10.6**	**0**	tl
summary	11.7	43	tl	11.7	40	tl	**11.4**	**48**	tl	11.5	43	tl

successful approach. For AIGS,B,UR the number of refinements per iteration is lower which results in a higher number of iterations. 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 computations 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 the performance. However, the quality of the bucket refinement is also very important.

Table 7.6: Comparison of ACPATHS, CPATH, and LUSED in combination with VDUE, SET+B and CPR. We consider the average optimality gaps (gap), the number of solved instances (opt) and the median computation times in seconds (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. The best values per instance set are highlighted bold.

	VDUE+ACPATHS SET+B CPR			VDUE+CPATH SET+B CPR			VDUE+LUSED SET+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**	14	**0.0**	**15**	**13**	**0.0**	**15**	28	**0.0**	**15**	**13**
e_{30}	**0.0**	**15**	71	0.6	14	74	0.6	14	**54**	0.6	14	66
e_{40}	**2.9**	**10**	215	4.4	**10**	179	5.0	**10**	186	4.8	**10**	**160**
e_{50}	**0.0**	**15**	**105**	1.1	14	128	1.1	14	119	**0.0**	**15**	**105**
e_{60}	1.8	13	69	1.8	13	113	**1.2**	13	**58**	**1.2**	**14**	78
e_{70}	1.7	9	2301	0.9	13	1343	2.3	7	tl	**0.3**	**14**	**528**
e_{80}	0.6	11	1834	0.8	12	294	0.6	11	**190**	**0.4**	**13**	266
e_{90}	1.5	8	5746	**1.1**	**10**	**1830**	1.7	6	tl	1.5	9	1908
e_{100}	1.4	7	tl	1.4	7	tl	**1.2**	6	tl	**1.2**	**8**	**4740**
summary	**1.1**	103	215	1.3	108	179	1.5	96	186	**1.1**	**112**	**160**
h_{20}	2.3	14	20	**0.0**	**15**	25	**0.0**	**15**	**16**	**0.0**	**15**	20
h_{30}	8.9	9	1550	6.8	**12**	**839**	8.6	10	4344	**6.4**	**12**	985
h_{40}	**6.8**	**7**	tl	8.5	6	tl	11.9	5	tl	8.1	**7**	tl
h_{50}	18.9	3	tl	**18.2**	**4**	tl	**18.2**	3	tl	**18.2**	**4**	tl
h_{60}	16.2	2	tl	16.3	2	tl	16.8	2	tl	**15.3**	**4**	tl
h_{70}	20.5	0	tl	20.1	1	tl	21.0	0	tl	**19.9**	**3**	tl
h_{80}	14.6	1	tl	14.5	1	tl	14.6	1	tl	**14.3**	**2**	tl
h_{90}	**9.4**	**1**	tl	9.9	**1**	tl	9.9	**1**	tl	10.0	**1**	tl
h_{100}	**10.6**	**0**	tl	**10.6**	**0**	tl	**10.6**	**0**	tl	**10.6**	**0**	tl
summary	12.0	37	tl	11.7	42	tl	12.4	37	tl	**11.4**	**48**	tl

For an illustration see Figure 7.1. 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.

In Figure 7.2 we also compared all investigated refinement strategies in a pairwise fashion checking the assumption that one strategy yields smaller 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. The Wilcoxon test coincides with our evaluation with one exception. Considering the optimality gaps only, VDUE,B,CPR performs best for the easy instance sets. However, when also considering the number of solved instances, VDUE,SET+B,MPR performs best for the easy instance sets, as the strategy solves the highest number of easy instances and yields only non-significantly worse gaps than VDUE,B,CPR. The bucket selection strategies AIGS, ASEL and ISSC are vastly outperformed by VDUE. In total VDUE,SET+B,CPR and VDUE,SET+B,MPR provide the best gaps.

Table 7.7: Comparison of ACPATHS, CPATH, and LUSED in combination with VDUE, SET+B and MPR. We consider the average optimality gaps (gap), the number of solved instances (opt) and the median computation times in seconds (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. The best values per instance set are highlighted bold.

	VDUE+ACPATHS SET+B MPR			VDUE+CPATH SET+B MPR			VDUE+LUSED SET+B MPR			VDUE SET+B MPR		
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**	11	**0.0**	**15**	**10**	**0.0**	**15**	14
e_{30}	0.6	14	59	0.6	14	70	0.6	14	51	**0.0**	**15**	**49**
e_{40}	**4.3**	9	**138**	5.0	9	262	4.6	9	514	4.5	**11**	281
e_{50}	1.1	14	116	**0.0**	**15**	138	1.1	14	182	**0.0**	**15**	**78**
e_{60}	1.7	13	**65**	**1.2**	**14**	77	1.8	12	75	**1.2**	**14**	75
e_{70}	1.4	11	1885	0.9	**12**	635	1.9	9	3105	**0.6**	11	**460**
e_{80}	0.9	10	1178	0.6	11	**607**	1.1	9	3570	**0.5**	**13**	1081
e_{90}	1.4	9	4397	1.7	7	tl	1.7	8	2264	**1.3**	**10**	**1835**
e_{100}	**1.0**	7	tl	1.2	**9**	**492**	1.4	7	tl	1.1	**9**	2845
summary	1.4	102	**138**	1.2	106	262	1.6	97	514	**1.0**	**113**	281
h_{20}	2.3	14	**15**	**0.0**	**15**	**15**	**0.0**	**15**	16	**0.0**	**15**	24
h_{30}	8.6	10	712	6.4	**12**	**568**	6.9	9	784	**6.3**	**12**	720
h_{40}	6.5	**7**	tl	**6.0**	**7**	tl	8.7	6	tl	9.1	6	tl
h_{50}	18.9	3	tl	**18.0**	**4**	tl	18.9	3	tl	18.2	**4**	tl
h_{60}	16.6	2	tl	17.0	2	tl	**14.9**	**3**	tl	15.5	2	tl
h_{70}	21.5	0	tl	21.9	0	tl	21.9	1	tl	**19.8**	**2**	tl
h_{80}	14.6	**1**	tl	14.6	**1**	tl	14.6	**1**	tl	**14.2**	**1**	tl
h_{90}	**9.5**	1	tl	9.7	**2**	tl	10.2	1	tl	10.1	1	tl
h_{100}	**10.6**	**0**	tl	**10.6**	**0**	tl	**10.6**	**0**	tl	**10.6**	**0**	tl
summary	12.1	38	tl	11.6	**43**	tl	11.9	39	tl	**11.5**	**43**	tl

Figure 7.3 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. Refinement strategies using ACAPTHS, CPATH and LUSED are part of this category. One can clearly see that these strategies generate much more buckets in usually fewer iterations than their counterparts without additional bucket selection strategies. Observe that the ASEL, ISCC, and AIGS variants are all located in the left upper half of the figure due to the large number of bucket splits they apply. 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. This is also true for SET+LSEE and LSEE. Note that 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.

Table 7.8: 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 (*ratio*$^{\mathrm{B}}$), the average number of iterations (n^{it}), and the average computation time spent per iteration in seconds ($t^{\mathrm{it}}[s]$). 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. The best values per instance set are highlighted bold.

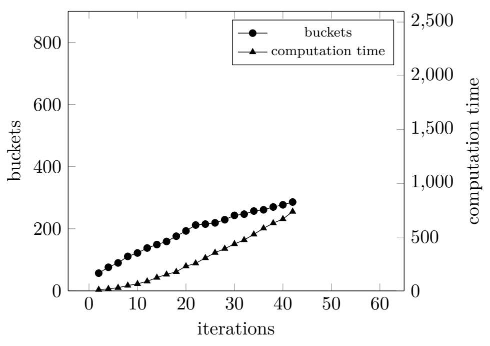
		ASEL B UR			AIGS B UR			VDUE B CPR			VDUE SET+B CPR		
set	$\lvert B^{\mathrm{init}}\rvert$	ratio$^{\mathrm{B}}$	n^{it}	$t^{\mathrm{it}}[s]$	ratio$^{\mathrm{B}}$	n^{it}	$t^{\mathrm{it}}[s]$	ratio$^{\mathrm{B}}$	n^{it}	$t^{\mathrm{it}}[s]$	ratio$^{\mathrm{B}}$	n^{it}	$t^{\mathrm{it}}[s]$
e_{20}	43	4.73	**9**	5	5.59	11	6	1.93	16	**2**	**1.69**	**9**	**2**
e_{30}	44	7.30	**9**	85	8.30	12	88	**2.89**	24	**17**	2.91	16	22
e_{40}	47	7.14	**7**	454	6.72	8	383	2.99	19	**158**	**2.91**	13	191
e_{50}	45	8.53	**7**	152	12.17	11	144	**2.83**	15	**25**	2.87	12	33
e_{60}	49	5.30	**4**	218	6.00	5	270	2.50	11	**72**	**2.30**	6	81
e_{70}	49	10.18	**6**	573	10.61	7	420	3.59	18	**89**	**3.52**	12	103
e_{80}	52	7.45	**4**	629	7.01	**4**	435	3.35	13	131	**3.23**	9	**118**
e_{90}	52	6.94	**3**	809	7.53	4	640	3.54	11	270	**3.53**	8	**235**
e_{100}	58	7.69	**3**	951	7.11	**3**	904	3.99	13	**266**	**3.75**	9	312
summary	48	7.25	**6**	431	7.89	7	366	3.07	16	**114**	**2.97**	10	122
h_{20}	43	4.25	**8**	8	5.55	12	11	**1.93**	17	**3**	2.00	13	**3**
h_{30}	44	6.84	**9**	418	7.90	11	388	**3.08**	23	121	3.37	19	**109**
h_{40}	43	6.98	**6**	924	7.84	7	770	3.39	17	**276**	**3.29**	12	313
h_{50}	48	5.27	**3**	1812	5.14	**3**	1529	**2.93**	11	**743**	3.06	9	820
h_{60}	44	5.49	**2**	1926	6.58	3	1728	**3.50**	11	**803**	3.59	9	855
h_{70}	48	4.86	**1**	2629	5.05	2	2446	**3.08**	7	**1280**	3.17	6	1332
h_{80}	49	4.94	**1**	2362	4.80	2	2162	**3.01**	7	**1043**	3.15	5	1112
h_{90}	54	4.97	**1**	2239	4.99	2	2087	**3.04**	6	**1017**	3.22	5	1161
h_{100}	55	4.96	**1**	2617	4.84	**1**	2433	**2.86**	4	**1287**	3.02	4	1430
summary	48	5.40	**4**	1659	5.85	5	1506	**2.98**	11	**730**	3.10	9	793

7.2.3 Comparing ITBRA to Other Algorithms

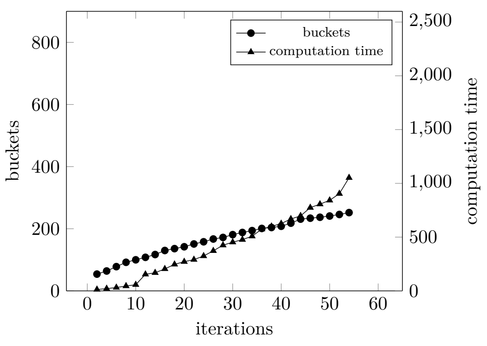
After having identified good refinement strategies for ITBRA, we can now proceed with the comparison to other algorithms. As ITBRA produces optimal solutions as well as heuristic solutions when it terminates prematurely, we compare our algorithm to exact methods and heuristics. We start by comparing the matheuristic to the stand-alone GRASP. Afterwards, we compare ITBRA to DEF and TIF. Finally, we compare our matheuristic with a heuristic based on TIF. For the comparisons we choose the refinement strategies VDUE,SET+B,MPR and VDUE,SET+B,CPR for ITBRA, as they produced the best results in our previous experiments.

GRASP

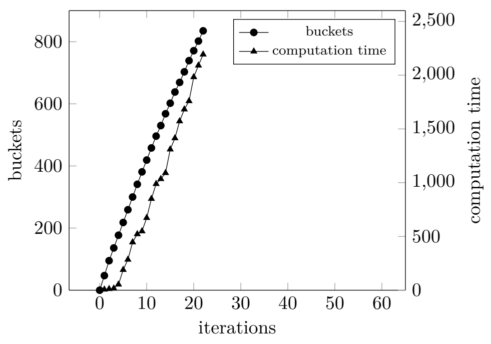
We start by comparing the matheuristic to the stand-alone GRASP, see Table 7.9. ITBRA is in general able to provide better results. However, when dealing with the most difficult instances, it is sometimes the case that the matheuristic only completes very few iterations and GRASP is able to compute a slightly better solution. As the number



(a) VDUE,SET+B,CPR



(b) VDUE,B,CPR



(c) ASEL,B,UR

Figure 7.1: 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.

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 dual bounds which GRASP cannot provide at all.

DEF

DEF was not able to find a primal solution for any instance but at least always computed a dual bound. Table 7.10 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; by the construction of the test instances, the activity time windows of the hard instances are significantly larger than the activity time windows of the easy instances, which results in larger domains of the starting time variables of DEF and thus also potentially larger big-M constraints.

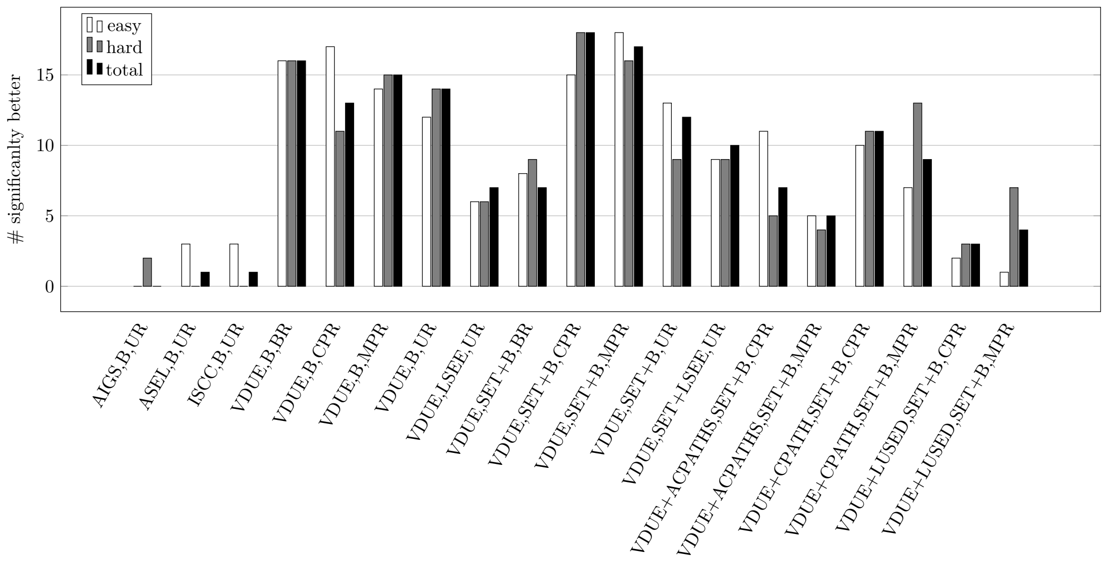


Figure 7.2: Comparison of refinement strategies using a one-tailed Wilcoxon rank-sum test with a significance level of 0.05 per difficulty setting and in total w.r.t. optimality gaps.

Time-Indexed Models

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 $T^\kappa = \{t \in T : t \equiv 0 \pmod{\kappa}\}$. 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 7.11 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 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

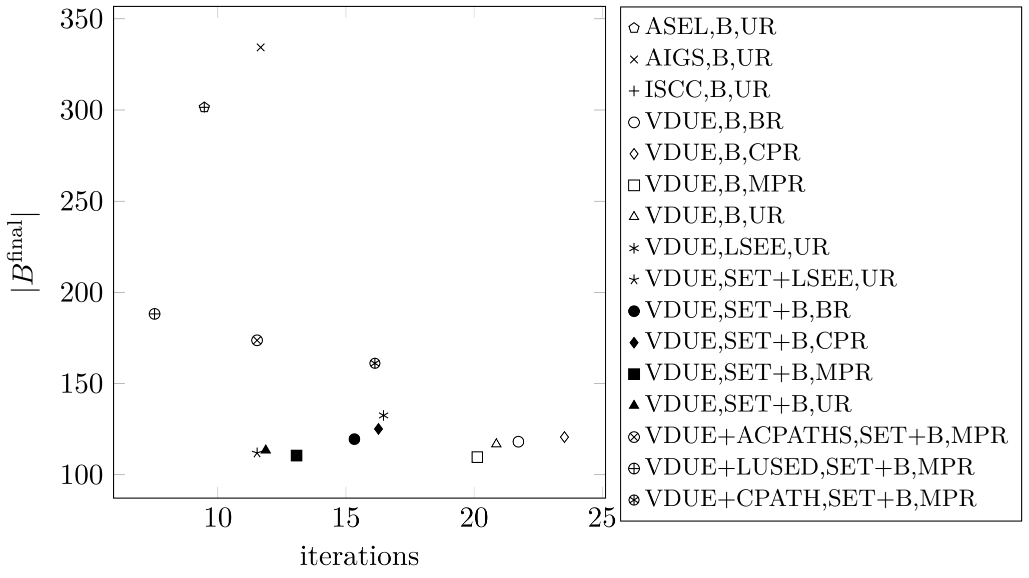


Figure 7.3: Comparison of the average number of iterations and average final number of buckets on the set of e_{30} instances.

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 80%. Therefore, further increasing the value of κ does not seem meaningful anymore. 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.

Comparing TIF to DTIF, STIF, and ITBRA

Models DTIF and TIF lead to similar memory consumption issues as the basic TIF model. Thus, we compare TIF^{κ} to $DTIF^{\kappa}$ and $STIF^{\kappa}$ in the following. Tables 7.12–7.16 show the results of these comparisons for different values of κ. For $\kappa \leq 200$, TIF^{κ} produces the best results as $DTIF^{\kappa}$ and $STIF^{\kappa}$ struggle with the time and memory limit.

Starting at $\kappa = 1000$, $DTIF^{1000}$ starts to catch up to TIF^{1000}. TIF^{1000} is still able to beat the results generated by ITBRA more often than $DTIF^{1000}$. However, $DTIF^{1000}$ is able

Table 7.9: 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 in seconds(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. The best values per instance set are highlighted bold.

	VDUE SET+B CPR			VDUE SET+B MPR			GRASP			
set	gap[%]	σ	$t[s]$	gap[%]	σ	$t[s]$	gap[%]	σ	$t[s]$	feas
e_{20}	**0.0**	**0.0**	**13**	**0.0**	**0.0**	14	38.6	17.3	tl	**12**
e_{30}	0.6	2.1	66	**0.0**	**0.0**	**49**	28.0	16.0	tl	**14**
e_{40}	4.6	7.7	**160**	**4.3**	7.6	281	13.1	**7.5**	tl	**15**
e_{50}	**0.0**	**0.0**	105	**0.0**	**0.0**	**78**	7.8	8.0	tl	**15**
e_{60}	**0.8**	**2.9**	78	**0.8**	**2.9**	**75**	3.4	4.6	tl	**15**
e_{70}	**0.3**	**1.0**	528	0.6	1.3	**460**	3.8	3.9	tl	**15**
e_{80}	**0.1**	**0.4**	**266**	0.2	0.8	1081	2.2	3.7	tl	**15**
e_{90}	0.8	1.2	1908	**0.6**	**1.1**	**1835**	1.0	1.2	tl	**15**
e_{100}	0.7	1.5	4740	**0.6**	**1.4**	**2845**	1.3	2.1	tl	**15**
summary	0.9	1.9	**160**	**0.8**	**1.7**	281	11.0	7.1	tl	**131**
h_{20}	**0.0**	**0.0**	**20**	**0.0**	**0.0**	24	23.3	12.7	tl	**12**
h_{30}	5.2	**11.4**	985	**5.0**	**11.4**	**720**	34.1	11.9	tl	**15**
h_{40}	**5.9**	**8.1**	tl	7.3	8.7	tl	21.6	11.6	tl	**15**
h_{50}	**8.9**	8.0	tl	**8.9**	8.0	tl	11.9	**6.7**	tl	**15**
h_{60}	**6.8**	6.9	tl	7.1	**6.3**	tl	9.5	6.5	tl	**15**
h_{70}	2.5	3.4	tl	**2.4**	**3.0**	tl	4.8	6.2	tl	**15**
h_{80}	1.4	**2.3**	tl	**1.3**	**2.3**	tl	1.6	3.1	tl	**15**
h_{90}	**2.4**	**4.3**	tl	2.5	4.8	tl	2.8	6.4	tl	**15**
h_{100}	1.0	0.9	tl	1.0	0.9	tl	**0.1**	**0.5**	tl	**15**
summary	**3.8**	**5.0**	tl	3.9	**5.0**	tl	12.2	7.3	tl	**132**

to find a higher number of feasible solution and generates slightly better gaps. *STIF*^{1000} is still far behind the other two algorithms w.r.t. the number of feasible solutions and the number of solved instances.

For $\kappa = 2000$, *DTIF*^{2000} starts to pull ahead of *TIF*^{2000}. For most instance sets *TIF*^{2000} and *DTIF*^{2000} perform almost equally well. However, for the instance sets h_{70}, h_{80}, and h_{90}, *DTIF*^{2000} significantly outperforms *TIF*^{2000}. *STIF*^{2000} is still not able to catch up to the other two algorithms. Moreover, for some instance sets *STIF*^{2000} still fails to produce feasible solutions due to the time and memory limit.

Finally, for $\kappa = 10000$, most instances have become infeasible and all three algorithms generate the same gaps for the few instances still permitting a feasible solution. However, even now *STIF*^{10000} struggles with the time and memory limit.

Table 7.10: Comparison of ITBRA with 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). “tl" indicates the termination of the program due to the time limit. The summary is obtained by aggregating over the preceding rows using the same function as for the respective column. The best values per instance set are highlighted bold.

	VDUE SET+B CPR			VDUE SET+B MPR			DEF		
set	gap[%]	σ	$t[s]$	gap[%]	σ	$t[s]$	gap[%]	σ	$t[s]$
e_{20}	**0.0**	**0.0**	**13**	**0.0**	**0.0**	14	15.3	10.6	tl
e_{30}	**0.0**	**0.0**	66	**0.0**	**0.0**	**49**	7.6	6.5	tl
e_{40}	0.9	2.7	**160**	**0.6**	**2.1**	281	5.0	9.3	tl
e_{50}	**0.0**	**0.0**	105	**0.0**	**0.0**	**78**	3.0	6.4	tl
e_{60}	**0.0**	**0.0**	78	**0.0**	**0.0**	**75**	1.8	3.2	tl
e_{70}	**0.0**	**0.0**	528	0.3	1.1	**460**	1.6	2.4	tl
e_{80}	**0.0**	**0.1**	**266**	**0.0**	**0.1**	1081	1.0	1.7	tl
e_{90}	0.1	0.2	1908	**0.0**	**0.0**	**1835**	1.5	2.8	tl
e_{100}	0.1	**0.1**	4740	**0.0**	**0.1**	**2845**	2.0	1.8	tl
summary	**0.1**	**0.3**	**160**	**0.1**	0.4	281	4.3	5.0	tl
h_{20}	**0.0**	**0.0**	**20**	**0.0**	**0.0**	24	19.6	11.4	tl
h_{30}	0.4	1.1	985	**0.2**	**0.6**	**720**	32.0	12.6	tl
h_{40}	**0.8**	**2.3**	tl	1.2	3.3	tl	19.2	12.8	tl
h_{50}	**2.3**	**2.7**	tl	2.5	3.1	tl	15.4	9.0	tl
h_{60}	2.1	4.4	tl	**1.9**	**3.9**	tl	5.2	5.9	tl
h_{70}	3.5	5.1	tl	**3.4**	**4.9**	tl	7.4	7.6	tl
h_{80}	**0.1**	0.4	tl	**0.1**	**0.3**	tl	1.0	1.6	tl
h_{90}	**0.8**	**1.7**	tl	**0.8**	**1.7**	tl	4.4	5.8	tl
h_{100}	0.3	0.6	tl	**0.2**	**0.3**	tl	1.0	1.5	tl
summary	**1.1**	**2.0**	tl	**1.1**	**2.0**	tl	11.7	7.6	tl

Similar to Figure 7.4, we compared all investigated time-indexed models using a one-tailed Wilcoxon rank-sum test with a significance level of 0.05 based on the gaps to the best found primal bounds of the instances. The results are shown in Figure 7.4 and coincide with our evaluation. *TIF*$^{\kappa}$ produces smaller models and has therefore an advantage over the other two algorithms for small values of κ. As κ increases, so does the performance of *DTIF*$^{\kappa}$. While *TIF*^{200} yields the best gaps for the easy instance sets, *DTIF*^{1000} turns out to be the best algorithm in general. *STIF*$^{\kappa}$ produces by far the worst results for high values of κ even though *STIF* is a stronger formulation than *TIF* and equally strong as DTIF. This can mostly be explained due to the larger size of the model of *STIF*.

Finally, in Figure 7.5 we compare a selection of ITBRA strategies, GRASP, and differently coarsened time-indexed models. The comparison is done by a one-tailed Wilcoxon rank-sum test with a significance level of 0.05 and is based on the gaps to the best found primal bounds of the instances. We have chosen the best as well as two inferior reference bucket refinement strategies for ITBRA and compare them to the best found time-indexed models and GRASP. The time-indexed models featured in Figure 7.5 are slightly

Table 7.11: 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 in seconds (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. The best values per instance set are highlighted bold.

	TIF^{100}		TIF^{200}		TIF^{1000}		TIF^{2000}		TIF^{10000}		VDUE B CPR		VDUE SET+B CPR	
set	gap[%]$^{\text{med}}$	$t[s]$	gap[%]$^{\text{med}}$	$t[s]$	gap[%]$^{\text{med}}$	$t[s]$	gap[%]$^{\text{med}}$	$t[s]$	gap[%]$^{\text{med}}$	$t[s]$	gap[%]$^{\text{med}}$	$t[s]$	gap[%]$^{\text{med}}$	$t[s]$
e_{20}	0.3	15	0.6	4	21.7	**<1**	24.0	**<1**	-	-	**0.0**	13	**0.0**	14
e_{30}	0.1	310	0.4	58	4.0	4	18.3	**2**	-	-	**0.0**	66	**0.0**	49
e_{40}	0.4	2940	0.5	698	3.8	40	13.5	**6**	-	-	**0.0**	160	**0.0**	281
e_{50}	0.2	409	0.4	113	1.9	17	6.4	5	-	**<1**	**0.0**	105	**0.0**	78
e_{60}	0.3	5569	0.4	839	2.4	52	4.7	21	-	**<1**	**0.0**	78	**0.0**	75
e_{70}	14.3	tl	0.3	2713	1.9	165	4.7	77	-	**<1**	**0.0**	528	**0.0**	460
e_{80}	-	tl	0.3	5630	1.7	292	3.4	108	-	**<1**	**0.0**	266	**0.0**	1081
e_{90}	-	tl	-	tl	1.1	555	3.0	327	-	**1**	**0.0**	1908	**0.0**	1835
e_{100}	-	tl	-	tl	1.8	652	4.0	263	-	**6**	**0.0**	4740	**0.0**	2845
summary	0.4	5569	0.4	839	1.9	52	4.7	21	-	**<1**	**0.0**	160	**0.0**	281
h_{20}	0.4	39	0.5	8	6.4	1	19.8	**<1**	-	**<1**	**0.0**	20	**0.0**	24
h_{30}	11.8	6106	0.6	1129	11.1	42	24.6	**13**	-	-	**0.0**	985	**0.0**	720
h_{40}	35.4	tl	**0.6**	tl	5.5	227	13.7	65	-	**<1**	3.2	tl	6.5	tl
h_{50}	-	tl	-	tl	**2.6**	2815	11.3	381	-	**<1**	8.4	tl	8.4	tl
h_{60}	-	tl	8.9	tl	**2.5**	1532	9.3	940	-	**<1**	6.5	tl	6.2	tl
h_{70}	-	tl	-	tl	14.3	tl	10.4	3052	82.5	**<1**	**0.0**	tl	0.7	tl
h_{80}	-	tl	-	tl	5.8	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**	**0.3**	tl	0.7	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**

Table 7.12: Comparison of different time-indexed models for $\kappa = 100$. We provide the median gaps to the best primal bound of the original problem (gap) and the median computation times in seconds (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. We also provide the number of instances per instances set for which a time-indexed model has found a better solution than ITBRA (better). 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. The best values per instance set are highlighted bold.

	TIF^{100}		DTIF^{100}		STIF^{100}	
set	gap[%]^{med}	$t[s]$	gap[%]^{med}	$t[s]$	gap[%]^{med}	$t[s]$
e_{20}	**0.3**	**15**	**0.3**	64	**0.3**	279
e_{30}	**0.1**	**310**	-	tl	10.9	tl
e_{40}	**0.4**	**2940**	-	tl	92.7	tl
e_{50}	**0.2**	**409**	-	tl	79.4	tl
e_{60}	**0.3**	**5569**	-	tl	122.9	tl
e_{70}	**14.3**	tl	-	-	-	tl
e_{80}	**-**	tl	**-**	-	**-**	tl
e_{90}	**-**	tl	**-**	-	**-**	tl
e_{100}	**-**	tl	**-**	-	**-**	tl
summary	**0.4**	**5569**	-	tl	122.9	tl
h_{20}	**0.4**	**39**	**0.4**	185	**0.4**	429
h_{30}	**11.8**	**6106**	-	tl	36.8	tl
h_{40}	**35.4**	tl	-	tl	192.8	tl
h_{50}	**-**	tl	**-**	tl	**-**	tl
h_{60}	**-**	tl	**-**	-	**-**	tl
h_{70}	**-**	tl	**-**	-	**-**	tl
h_{80}	**-**	tl	**-**	-	**-**	tl
h_{90}	**-**	tl	**-**	-	**-**	tl
h_{100}	**-**	tl	**-**	-	**-**	tl
summary	**-**	tl	**-**	tl	**-**	tl

	TIF^{100}					DTIF^{100}					STIF^{100}				
set	opt	feas	infeas	better	mem	opt	feas	infeas	better	mem	opt	feas	infeas	better	mem
e_{20}	**15**	**15**	0	**0**	**0**	**15**	**15**	0	**0**	**0**	**15**	**15**	0	**0**	**0**
e_{30}	**15**	**15**	0	**0**	**0**	2	2	0	**0**	13	4	**15**	0	**0**	**0**
e_{40}	**9**	12	0	**0**	**0**	0	0	0	**0**	15	0	**13**	0	**0**	**0**
e_{50}	**13**	**14**	0	**0**	**0**	0	0	0	**0**	15	0	11	0	**0**	**0**
e_{60}	**9**	**9**	0	**0**	**0**	0	0	0	**0**	15	0	8	0	**0**	**0**
e_{70}	**6**	**9**	0	**0**	**0**	0	0	0	**0**	15	0	2	0	**0**	**0**
e_{80}	**3**	**3**	0	**0**	**0**	0	0	0	**0**	15	0	0	0	**0**	12
e_{90}	**0**	**0**	0	**0**	**2**	**0**	**0**	0	**0**	14	**0**	**0**	0	**0**	15
e_{100}	**1**	**1**	0	**0**	**6**	0	0	0	**0**	14	0	0	0	**0**	15
summary	**71**	**78**	0	**0**	**8**	17	17	0	**0**	116	19	64	0	**0**	42
h_{20}	**15**	**15**	0	**0**	**0**	**15**	**15**	0	**0**	**0**	**15**	**15**	0	**0**	**0**
h_{30}	**8**	12	0	**1**	**0**	1	1	0	0	14	2	**14**	0	0	**0**
h_{40}	**4**	**9**	0	**1**	**0**	0	0	0	0	15	0	8	0	0	**0**
h_{50}	**1**	**4**	0	**0**	**0**	0	0	0	**0**	15	0	1	0	**0**	**0**
h_{60}	**0**	0	0	**0**	**0**	**0**	0	0	**0**	15	**0**	**1**	0	**0**	**0**
h_{70}	**0**	**0**	0	**0**	**0**	**0**	**0**	0	**0**	15	**0**	**0**	0	**0**	14
h_{80}	**0**	**0**	0	**0**	**4**	**0**	**0**	0	**0**	14	**0**	**0**	0	**0**	15
h_{90}	**0**	**0**	0	**0**	**3**	**0**	**0**	0	**0**	13	**0**	**0**	0	**0**	15
h_{100}	**0**	**0**	0	**0**	**14**	**0**	**0**	0	**0**	15	**0**	**0**	0	**0**	**14**
summary	**28**	**40**	0	**2**	**21**	16	16	0	0	116	17	39	0	0	58

Table 7.13: Comparison of different time-indexed models for $\kappa = 200$. We provide the median gaps to the best primal bound of the original problem (gap) and the median computation times in seconds (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. We also provide the number of instances per instances set for which a time-indexed model has found a better solution than ITBRA (better). 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. The best values per instance set are highlighted bold.

	TIF^{200}		DTIF^{200}		STIF^{200}	
set	gap[%]$^{\text{med}}$	$t[s]$	gap[%]$^{\text{med}}$	$t[s]$	gap[%]$^{\text{med}}$	$t[s]$
e_{20}	**0.6**	**4**	**0.6**	16	**0.6**	46
e_{30}	**0.4**	**58**	**0.4**	225	**0.4**	1158
e_{40}	**0.5**	**698**	**0.5**	1403	14.7	tl
e_{50}	**0.4**	**113**	-	tl	78.8	tl
e_{60}	**0.4**	**839**	-	tl	78.1	tl
e_{70}	**0.3**	**2713**	-	tl	-	tl
e_{80}	**0.3**	**5630**	-	tl	-	tl
e_{90}	**-**	tl	**-**	tl	**-**	tl
e_{100}	**-**	tl	**-**	tl	**-**	tl
summary	**0.4**	**839**	-	tl	78.8	tl
h_{20}	**0.5**	**8**	**0.5**	40	**0.5**	59
h_{30}	**0.6**	1129	1.4	**861**	1.6	2653
h_{40}	**0.6**	tl	-	tl	20.9	tl
h_{50}	**-**	tl	**-**	tl	**-**	tl
h_{60}	**8.9**	tl	-	tl	-	tl
h_{70}	**-**	tl	**-**	tl	**-**	tl
h_{80}	**-**	tl	**-**	tl	**-**	tl
h_{90}	**-**	tl	**-**	tl	**-**	tl
h_{100}	**-**	tl	**-**	-	**-**	tl
summary	**-**	tl	**-**	tl	**-**	tl

	TIF^{200}					DTIF^{200}					STIF^{200}				
set	opt	feas	infeas	better	mem	opt	feas	infeas	better	mem	opt	feas	infeas	better	mem
e_{20}	**15**	**15**	0	**0**	**0**	**15**	**15**	0	**0**	**0**	**15**	**15**	0	**0**	**0**
e_{30}	**15**	**15**	0	**0**	**0**	**15**	**15**	0	**0**	**0**	**15**	**15**	0	**0**	**0**
e_{40}	**12**	14	0	1	**0**	10	14	0	**2**	1	4	**15**	0	1	**0**
e_{50}	**14**	**14**	0	**0**	**0**	1	1	0	**0**	14	1	9	0	**0**	**0**
e_{60}	**14**	**15**	0	**1**	**0**	0	0	0	0	15	0	9	0	0	**0**
e_{70}	**12**	**14**	0	**0**	**0**	0	0	0	**0**	15	0	4	0	**0**	**0**
e_{80}	**9**	**11**	0	**0**	**0**	0	0	0	**0**	15	0	0	0	**0**	11
e_{90}	**6**	**7**	0	**0**	**0**	0	0	0	**0**	15	0	0	0	**0**	15
e_{100}	**4**	**5**	0	**0**	**0**	0	0	0	**0**	15	0	0	0	**0**	15
summary	**101**	**110**	0	**2**	**0**	41	45	0	**2**	90	35	67	0	1	41
h_{20}	**15**	**15**	0	**0**	**0**	**15**	**15**	0	**0**	**0**	**15**	**15**	0	**0**	**0**
h_{30}	**14**	**15**	0	**3**	**0**	12	**15**	0	**3**	**0**	11	**15**	0	**3**	**0**
h_{40}	**8**	**12**	0	**5**	**0**	1	1	0	0	14	3	11	0	0	**0**
h_{50}	**4**	**7**	0	**1**	**0**	0	0	0	0	13	0	2	0	0	**0**
h_{60}	**2**	**9**	0	**4**	**0**	0	0	0	0	15	0	0	0	0	**0**
h_{70}	**0**	**1**	0	**0**	**0**	**0**	0	0	**0**	15	**0**	0	0	**0**	11
h_{80}	**1**	**4**	0	**0**	**0**	0	0	0	**0**	15	0	0	0	**0**	15
h_{90}	**1**	**1**	0	**0**	**0**	0	0	0	**0**	15	0	0	0	**0**	15
h_{100}	**0**	**0**	0	**0**	**0**	**0**	**0**	0	**0**	13	**0**	**0**	0	**0**	14
summary	**45**	**64**	0	**13**	**0**	28	31	0	3	100	29	43	0	3	55

Table 7.14: Comparison of different time-indexed models for $\kappa = 1000$. We provide the median gaps to the best primal bound of the original problem (gap) and the median computation times in seconds (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. We also provide the number of instances per instances set for which a time-indexed model has found a better solution than ITBRA (better). 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. The best values per instance set are highlighted bold.

	TIF^{1000}		DTIF^{1000}		STIF^{1000}	
set	gap[%]$^{\text{med}}$	$t[s]$	gap[%]$^{\text{med}}$	$t[s]$	gap[%]$^{\text{med}}$	$t[s]$
e_{20}	**21.7**	**<1**	**21.7**	1	**21.7**	4
e_{30}	**4.0**	**4**	**4.0**	6	**4.0**	16
e_{40}	**3.8**	40	**3.8**	**30**	**3.8**	70
e_{50}	**1.9**	**17**	**1.9**	20	**1.9**	318
e_{60}	**2.4**	**52**	**2.4**	67	**2.4**	892
e_{70}	**1.9**	**165**	**1.9**	215	2.0	3326
e_{80}	**1.7**	**292**	**1.7**	443	1.9	6350
e_{90}	**1.1**	**555**	**1.1**	885	-	tl
e_{100}	**1.8**	**652**	**1.8**	1160	-	tl
summary	**1.9**	**52**	**1.9**	67	3.8	892
h_{20}	**6.4**	**1**	**6.4**	**1**	**6.4**	5
h_{30}	**11.1**	42	**11.1**	**19**	**11.1**	39
h_{40}	**5.5**	227	**5.5**	**134**	**5.5**	274
h_{50}	2.6	2815	**0.0**	**1291**	**0.0**	1536
h_{60}	2.5	**1532**	**1.3**	3496	3.2	2582
h_{70}	14.3	tl	**13.4**	tl	26.0	tl
h_{80}	**5.8**	tl	10.8	tl	-	tl
h_{90}	**9.0**	tl	20.1	tl	-	tl
h_{100}	39.6	tl	**35.7**	tl	-	tl
summary	**6.4**	2815	10.8	3496	11.1	**2582**

	TIF^{1000}					DTIF^{1000}					STIF^{1000}				
set	opt	feas	infeas	better	mem	opt	feas	infeas	better	mem	opt	feas	infeas	better	mem
e_{20}	**15**	**15**	0	**0**	**0**	**15**	**15**	0	**0**	**0**	**15**	**15**	0	**0**	**0**
e_{30}	**15**	**15**	0	**0**	**0**	**15**	**15**	0	**0**	**0**	**15**	**15**	0	**0**	**0**
e_{40}	**15**	**15**	0	**1**	**0**	**15**	**15**	0	**1**	**0**	**15**	**15**	0	**1**	**0**
e_{50}	**15**	**15**	0	**0**	**0**	**15**	**15**	0	**0**	**0**	**15**	**15**	0	**0**	**0**
e_{60}	**15**	**15**	0	**1**	**0**	**15**	**15**	0	**1**	**0**	**15**	**15**	0	**1**	**0**
e_{70}	**15**	**15**	0	**0**	**0**	**15**	**15**	0	**0**	**0**	13	**15**	0	**0**	**0**
e_{80}	**15**	**15**	0	**0**	**0**	**15**	**15**	0	**0**	**0**	10	13	0	**0**	1
e_{90}	14	14	0	**2**	**0**	**15**	**15**	0	**2**	**0**	0	0	0	0	15
e_{100}	13	**15**	0	**0**	**0**	**14**	**15**	0	**0**	**0**	0	0	0	**0**	15
summary	132	134	0	**4**	**0**	**134**	**135**	0	**4**	**0**	98	103	0	2	31
h_{20}	**15**	**15**	0	**0**	**0**	**15**	**15**	0	**0**	**0**	**15**	**15**	0	**0**	**0**
h_{30}	**15**	**15**	0	**3**	**0**	**15**	**15**	0	**3**	**0**	**15**	**15**	0	**3**	**0**
h_{40}	**15**	**15**	0	**5**	**0**	**15**	**15**	0	**5**	**0**	**15**	**15**	0	**5**	**0**
h_{50}	12	**15**	0	**8**	**0**	14	**15**	0	**8**	**0**	**15**	**15**	0	**8**	**0**
h_{60}	**9**	**15**	0	7	**0**	**9**	**15**	0	**8**	**0**	**9**	**15**	0	6	**0**
h_{70}	**4**	10	0	**3**	**0**	2	**15**	0	1	**0**	0	8	0	**3**	4
h_{80}	**5**	11	0	**2**	**0**	4	**15**	0	1	**0**	0	0	0	0	15
h_{90}	**4**	12	0	**1**	**0**	**4**	**15**	0	**1**	**0**	0	0	0	0	15
h_{100}	**1**	8	0	**1**	**0**	**1**	**15**	0	**1**	**0**	0	0	0	0	15
summary	**80**	116	0	**30**	**0**	79	**135**	0	28	**0**	69	83	0	25	49

Table 7.15: Comparison of different time-indexed models for $\kappa = 2000$. We provide the median gaps to the best primal bound of the original problem (gap) and the median computation times in seconds (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. We also provide the number of instances per instances set for which a time-indexed model has found a better solution than ITBRA (better). 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. The best values per instance set are highlighted bold.

	TIF^{2000}		DTIF^{2000}		STIF^{2000}	
set	gap[%]^{med}	$t[s]$	gap[%]^{med}	$t[s]$	gap[%]^{med}	$t[s]$
e_{20}	**24.0**	<1	**24.0**	<1	**24.0**	3
e_{30}	**18.3**	**2**	**18.3**	**2**	**18.3**	8
e_{40}	**13.5**	**6**	**13.5**	11	**13.5**	16
e_{50}	**6.4**	**5**	**6.4**	7	**6.4**	36
e_{60}	**4.7**	21	**4.7**	**20**	**4.7**	84
e_{70}	**4.7**	**77**	**4.7**	89	**4.7**	341
e_{80}	**3.4**	**108**	**3.4**	119	**3.4**	584
e_{90}	3.0	327	**2.9**	**177**	-	tl
e_{100}	**4.0**	263	**4.0**	**208**	-	tl
summary	**4.7**	21	**4.7**	**20**	13.5	84
h_{20}	**19.8**	<1	**19.8**	<1	**19.8**	3
h_{30}	**24.6**	13	**24.6**	**5**	**24.6**	15
h_{40}	**13.7**	65	**13.7**	**22**	**13.7**	39
h_{50}	**11.3**	381	**11.3**	**136**	**11.3**	241
h_{60}	**9.3**	940	**9.3**	**359**	**9.3**	770
h_{70}	10.4	3052	**3.8**	2555	5.7	**2067**
h_{80}	12.1	tl	**5.2**	**6823**	-	tl
h_{90}	14.2	tl	**8.1**	tl	-	tl
h_{100}	**22.7**	tl	25.7	tl	-	tl
summary	13.7	940	**11.3**	**359**	19.8	770

	TIF^{2000}					DTIF^{2000}					STIF^{2000}				
set	opt	feas	infeas	better	mem	opt	feas	infeas	better	mem	opt	feas	infeas	better	mem
e_{20}	**13**	**13**	2	**0**	**0**	**13**	**13**	2	**0**	**0**	**13**	**13**	2	**0**	**0**
e_{30}	**15**	**15**	0	**0**	**0**	**15**	**15**	0	**0**	**0**	**15**	**15**	0	**0**	**0**
e_{40}	**15**	**15**	0	**0**	**0**	**15**	**15**	0	**0**	**0**	**15**	**15**	0	**0**	**0**
e_{50}	**15**	**15**	0	**0**	**0**	**15**	**15**	0	**0**	**0**	**15**	**15**	0	**0**	**0**
e_{60}	**15**	**15**	0	**0**	**0**	**15**	**15**	0	**0**	**0**	**15**	**15**	0	**0**	**0**
e_{70}	**15**	**15**	0	**0**	**0**	**15**	**15**	0	**0**	**0**	**15**	**15**	0	**0**	**0**
e_{80}	**15**	**15**	0	**0**	**0**	**15**	**15**	0	**0**	**0**	**15**	**15**	0	**0**	**0**
e_{90}	14	**15**	0	**0**	**0**	**15**	**15**	0	**0**	**0**	0	0	0	**0**	15
e_{100}	**15**	**15**	0	**0**	**0**	**15**	**15**	0	**0**	**0**	0	0	0	**0**	13
summary	132	**133**	2	**0**	**0**	**133**	**133**	2	**0**	**0**	103	103	2	**0**	28
h_{20}	**15**	**15**	0	**0**	**0**	**15**	**15**	0	**0**	**0**	**15**	**15**	0	**0**	**0**
h_{30}	**14**	**14**	1	**1**	**0**	**14**	**14**	1	**1**	**0**	**14**	**14**	1	**1**	**0**
h_{40}	**15**	**15**	0	**1**	**0**	**15**	**15**	0	**1**	**0**	**15**	**15**	0	**1**	**0**
h_{50}	**15**	**15**	0	**5**	**0**	**15**	**15**	0	**5**	**0**	**15**	**15**	0	**5**	**0**
h_{60}	14	**15**	0	**5**	**0**	14	**15**	0	**5**	**0**	**15**	**15**	0	**5**	**0**
h_{70}	10	**15**	0	0	**0**	**12**	**15**	0	**1**	**0**	11	**15**	0	**1**	**0**
h_{80}	4	**15**	0	0	**0**	**9**	**15**	0	**2**	**0**	0	0	0	0	14
h_{90}	6	12	0	**1**	**0**	**7**	**15**	0	**1**	**0**	0	0	0	0	15
h_{100}	0	11	0	0	**0**	**1**	**15**	0	**1**	**0**	0	0	0	0	15
summary	93	127	1	13	**0**	**102**	**134**	1	**17**	**0**	85	89	1	13	44

Table 7.16: Comparison of different time-indexed models for $\kappa = 10000$. We provide the median gaps to the best primal bound of the original problem (gap) and the median computation times in seconds (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. We also provide the number of instances per instances set for which a time-indexed model has found a better solution than ITBRA (better). 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. The best values per instance set are highlighted bold.

	TIF^{10000}		DTIF^{10000}		STIF^{10000}	
set	gap[%]^{med}	$t[s]$	gap[%]^{med}	$t[s]$	gap[%]^{med}	$t[s]$
e_{20}	**-**	**-**	**-**	**-**	**-**	**-**
e_{30}	**-**	**-**	**-**	**-**	**-**	**-**
e_{40}	**-**	**-**	**-**	**-**	**-**	**-**
e_{50}	**-**	**<1**	**-**	**<1**	**-**	10
e_{60}	**-**	**<1**	**-**	**<1**	**-**	15
e_{70}	**-**	**<1**	**-**	**<1**	**-**	17
e_{80}	**-**	**<1**	**-**	**<1**	**-**	22
e_{90}	**-**	1	**-**	**<1**	**-**	28
e_{100}	**-**	6	**-**	**1**	**-**	tl
summary	**-**	**<1**	**-**	**<1**	**-**	22
h_{20}	**-**	**<1**	**-**	**<1**	**-**	2
h_{30}	**-**	**-**	**-**	**-**	**-**	**-**
h_{40}	**-**	**<1**	**-**	**<1**	**-**	7
h_{50}	**-**	**<1**	**-**	**<1**	**-**	14
h_{60}	**-**	**<1**	**-**	**<1**	**-**	18
h_{70}	**82.5**	**<1**	**82.5**	**<1**	**82.5**	23
h_{80}	**77.0**	3	**77.0**	**<1**	**77.0**	31
h_{90}	**93.8**	8	**93.8**	**<1**	**93.8**	35
h_{100}	**-**	16	**-**	**1**	**-**	tl
summary	**-**	**<1**	**-**	**<1**	**-**	23

	TIF^{10000}					DTIF^{10000}					STIF^{10000}				
set	opt	feas	infeas	better	mem	opt	feas	infeas	better	mem	opt	feas	infeas	better	mem
e_{20}	**0**	**0**	15	**0**	**0**	**0**	**0**	15	**0**	**0**	**0**	**0**	15	**0**	**0**
e_{30}	**0**	**0**	15	**0**	**0**	**0**	**0**	15	**0**	**0**	**0**	**0**	15	**0**	**0**
e_{40}	**0**	**0**	15	**0**	**0**	**0**	**0**	15	**0**	**0**	**0**	**0**	15	**0**	**0**
e_{50}	**2**	**2**	13	**0**	**0**	**2**	**2**	13	**0**	**0**	**2**	**2**	13	**0**	**0**
e_{60}	**6**	**6**	9	**0**	**0**	**6**	**6**	9	**0**	**0**	**6**	**6**	9	**0**	**0**
e_{70}	**2**	**2**	13	**0**	**0**	**2**	**2**	13	**0**	**0**	**2**	**2**	13	**0**	**0**
e_{80}	**7**	**7**	8	**0**	**0**	**7**	**7**	8	**0**	**0**	**7**	**7**	8	**0**	**0**
e_{90}	**4**	**4**	11	**0**	**0**	**4**	**4**	11	**0**	**0**	**4**	**4**	11	**0**	**0**
e_{100}	**2**	**2**	13	**0**	**0**	**2**	**2**	13	**0**	**0**	0	0	12	**0**	3
summary	**23**	**23**	112	**0**	**0**	**23**	**23**	112	**0**	**0**	21	21	111	**0**	3
h_{20}	**2**	**2**	13	**0**	**0**	**2**	**2**	13	**0**	**0**	**2**	**2**	13	**0**	**0**
h_{30}	**0**	**0**	15	**0**	**0**	**0**	**0**	15	**0**	**0**	**0**	**0**	15	**0**	**0**
h_{40}	**5**	**5**	10	**0**	**0**	**5**	**5**	10	**0**	**0**	**5**	**5**	10	**0**	**0**
h_{50}	**6**	**6**	9	**0**	**0**	**6**	**6**	9	**0**	**0**	**6**	**6**	9	**0**	**0**
h_{60}	**6**	**6**	9	**0**	**0**	**6**	**6**	9	**0**	**0**	**6**	**6**	9	**0**	**0**
h_{70}	**8**	**8**	7	**0**	**0**	**8**	**8**	7	**0**	**0**	**8**	**8**	7	**0**	**0**
h_{80}	**9**	**9**	6	**0**	**0**	**9**	**9**	6	**0**	**0**	**9**	**9**	6	**0**	**0**
h_{90}	**8**	**8**	7	**0**	**0**	**8**	**8**	7	**0**	**0**	**8**	**8**	7	**0**	**0**
h_{100}	**5**	**5**	10	**0**	**0**	**5**	**5**	10	**0**	**0**	0	0	0	**0**	15
summary	**49**	**49**	86	**0**	**0**	**49**	**49**	86	**0**	**0**	44	44	76	**0**	15

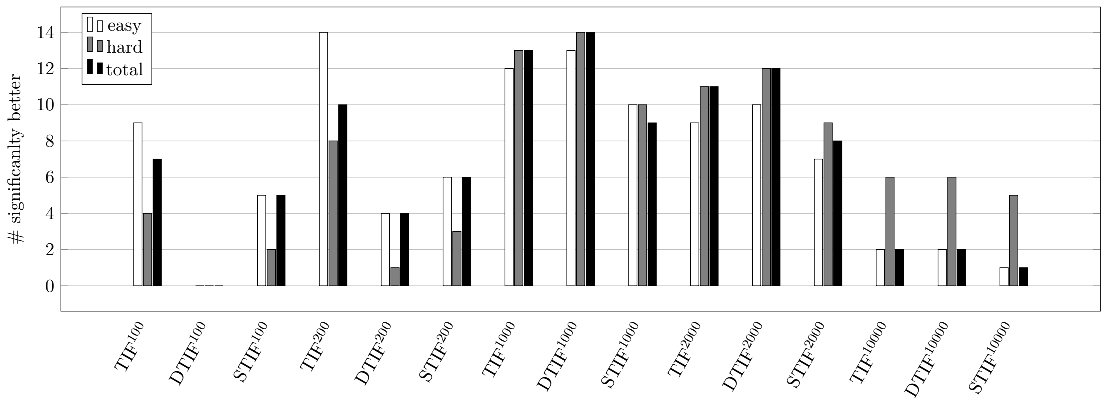


Figure 7.4: Comparison of time-indexed models using a one-tailed Wilcoxon rank-sum test with a significance level of 0.05 per difficulty setting and in total w.r.t. the gaps to the best found primal bounds.

superior to GRASP especially w.r.t. the easy instance sets. Tables 7.12–7.16 show that the time-indexed models are able to generate better primal bounds than ITBRA for some instance sets. However, in general, even the most simple ITBRA refinement strategies vastly outperform all time-indexed models w.r.t. generating primal bounds.

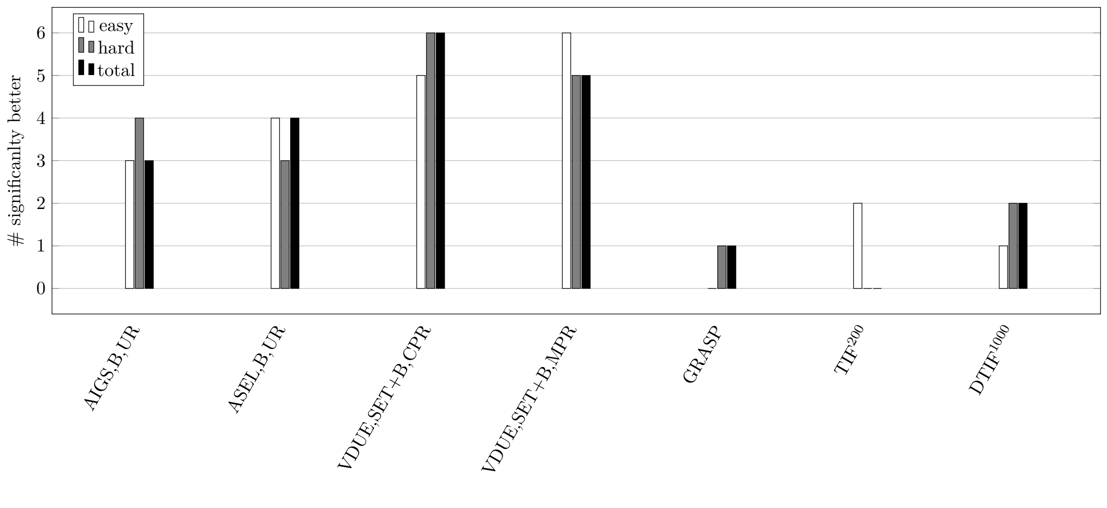


Figure 7.5: Comparison of a selection of algorithms using a one-tailed Wilcoxon rank-sum test with a significance level of 0.05 per difficulty setting and in total w.r.t. gaps to the best found primal bounds.

CHAPTER 8

Conclusion and Future Work

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. Strategies VDUE,SET+B,CPR and VDUE,SET+B,MPR 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) formulations. The considered discrete-event formulation (DEF) is only capable of computing dual bounds for all of our benchmark instances but no primal bounds 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.

The disaggregated time-indexed formulation (DTIF) and the time-indexed formulation with step variables (STIF) are also not able to keep up with ITBRA. STIF generates the largest models which drastically increases the memory consumption and computation time. For less coarsened time horizons DTIF struggles as well with the memory limit. However, the more coarsened the time horizon is, the better is the performance of DTIF. DTIF starts outperforming TIF shortly before the instances become infeasible due to the crude time horizon.

8.1 Future Work

We primarily focused on MILP-based algorithms here. Another well-known exact technique often used to deal with scheduling problems is constraint programming (CP). Consequently, 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 elaborated techniques should be identified and tested.

The bucket refinement strategies have a major impact on the total performance of ITBRA. While the bucket selection strategy VDUE is unarguably superior to the other bucket selection strategies, we could not identify a clear best strategy for the other parts of the refinement strategy. Hence, developing more elaborate refinement strategies might greatly improve the performance of ITBRA.

In this thesis bucket refinement strategies are only compared empirically. However, a theoretical comparison of the investigated bucket refinement is necessary in order to confirm our empirical evaluation and may also be useful for finding new refinement strategies. In the context of comparing bucket refinement strategies, deriving minimal bucket partitionings, i.e., bucket partitionings with a minimal number of buckets s.t. the time-bucket relaxation (TBR) yields a feasible (and therefore optimal) solution, may also be of interest. Even more interesting are minimum bucket partitionings, i.e., minimal refinements with the smallest number of buckets. However, computing (and proving) such refinements appears to be at least as challenging as finding optimal solutions.

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.

List of Figures

List of Tables

List of Algorithms

Acronyms

SI-PTPSP simplified intraday particle therapy patient scheduling problem

GCH gap closing heuristic

ABCH activity block construction heuristic

RCMPSP resource-constrained multi-project scheduling problem

MRCMPSP multi-mode resource-constrained multi-project scheduling problem

RCPSP resource-constrained project scheduling problem

TSPTW traveling salesman problem with time windows

MVRPTW vehicle routing problem with time windows and multiple routes

CTSNDP countinuous time service network design problem

LLB linear lower bounds

VNS variable neighborhood search

GA genetic algorithm

GRASP greedy randomized adaptive search procedure

CP constraint programming

LP linear programming

MILP mixed integer linear programming

ILP integer linear programming

BILP binary integer linear program

B&B branch-and-bound

B&C branch-and-cut

TIF	time-indexed formulation
DTIF	disaggregated time-indexed formulation
STIF	time-indexed formulation with step variables
DEF	discrete-event formulation
TBR	time-bucket relaxation
DTBR	disaggregated time-bucket relaxation
ETBR	extended time-bucket relaxation
ITBRA	iterative time-bucket refinement algorithm

Bibliography

C. Artigues. A note on time-indexed formulations for the resource-constrained project scheduling problem. *Operations Research Letters*, pages 1–15, 2013.

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 *Proceedings of the 6th Multidisciplinary International Scheduling Conference*, pages 815–819, Ghent, Belgium, 27–30 Aug. 2013.

C. Artigues, S. Demassey, and E. Neron. *Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications.* Wiley-ISTE, ISTE Ltd: 6 Fitzroy Square London W1T 5DX, John Wiley & Sons: 111 River Street Hoboken, NJ 07030 USA, 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.

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, Berlin, Heidelberg, 2010. Springer.

D. Bertsimas and J. N. Tsitsiklis. *Introduction to linear optimization*, volume 6. Athena Scientific Belmont, MA, 1997.

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, 2011.

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.

C. Blum and G. R. Raidl. *Hybrid Metaheuristics – Powerful Tools for Optimization*. Artificial Intelligence: Foundations, Theory, and Algorithms. Springer, 2016.

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.

F. Bomsdorf and U. Derigs. A model, heuristic procedure and decision support system for solving the movie shoot scheduling problem. *OR Spectrum*, 30(4):751–772, 2008.

J. Böttcher, A. Drexl, R. Kolisch, and F. Salewski. Project Scheduling Under Partially Renewable Resource Constraints. *Management Science*, 45(4):543–559, 1999.

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 and E. Néron. On linear lower bounds for the resource constrained project scheduling problem. *European Journal of Operational Research*, 149(2):314–324, 2003.

M. Caserta and S. Voß. *Metaheuristics: Intelligent Problem Solving*, pages 1–38. Springer US, Boston, MA, 2010.

C. Cavalcante, C. Carvalho De Souza, M. W P Savelsbergh, Y. Wang, and L. A. Wolsey. Scheduling projects with labor constraints. *Discrete Applied Mathematics*, 112(1-3): 27–52, 2001.

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.

A. Cesta, A. Oddi, and S. F. Smith. A Constraint-Based Method for Project Scheduling with Time Windows. *Journal of Heuristics*, 8(1):109–136, 2002.

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.

G. B. Dantzig. Maximization of a linear function of variables subject to linear inequalities. *New York*, 1951.

G. B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations research*, 8(1):101–111, 1960.

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.

B. De Reyck and W. Herroelen. A branch-and-bound procedure for the resource-constrained project scheduling problem with generalized precedence relations. *European Journal of Operational Research*, 111(1):152–174, 1998.

F. Della Croce, F. Salassa, and V. T'Kindt. A hybrid heuristic approach for single machine scheduling with release times. *Computers and Operations Research*, 45:7–11, 2014.

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*, pages 60–71, Brussels, Belgium, 4–7 Sep. 2016.

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

F. Glover. A multiphase-dual algorithm for the zero-one integer programming problem. *Operations Research*, 13(6):879–919, 1965.

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.

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.

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

N. Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311. ACM, 1984.

L. G. Khachiyan. A polynomial algorithm in linear programming. In *Doklady Akademiia Nauk SSSR*, volume 244, pages 1093–1096, 1979.

L. G. Khachiyan. Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1):53–72, 1980.

R. Klein. Project scheduling with time-varying resource constraints. *International Journal of Production Research*, 38(16):3937–3952, 2000.

R. Kolisch. *Project Scheduling under Resource Constraints.* Physica-Verlag HD, Heidelberg, 1995.

R. Kolisch and S. Hartmann. Experimental Investigation of Heuristics for Resource Constrained Project Scheduling: An Update. *European Journal of Operational Research*, 174(1):23–37, 2006.

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.

A. Levin. Scheduling and Fleet Routing Models for Transportation Systems. *Transportation Science*, 5(3):232–255, 1971.

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.

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.

C. H. Papadimitriou. On the complexity of integer programming. *Journal of the ACM (JACM)*, 28(4):765–768, 1981.

G. R. Raidl. Decomposition based hybrid metaheuristics. *European Journal of Operational Research*, 244(1):66–76, 2015.

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.

M. Riedler, T. Jatschka, J. Maschler, and G. R. Raidl. An iterative time-bucket refinement algorithm for a high-resolution resource-constrained project scheduling problem. *International Transactions in Operational Research*, to appear. available at `http://dx.doi.org/10.1111/itor.12445`.

A. Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, 1998.

A. J. Swersey and W. Ballard. Scheduling school buses. *Management Science*, 30(7): 844–853, 1984.

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.

L. A. Wolsey. *Integer programming*. Wiley, 1998.