FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Patient Scheduling in Particle Therapy

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Technischen Wissenschaften

by

Dipl.-Ing. Johannes Maschler, BSc
Registration Number 0725953

to the Faculty of Informatics
at the TU Wien
Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Günther Raidl

The dissertation has been reviewed by:

Luca Di Gaspero	Nysret Musliu	Günther R. Raidl

Vienna, 31st January, 2019

Johannes Maschler

Erklärung zur Verfassung der Arbeit

Dipl.-Ing. Johannes Maschler, BSc
Mollardgasse 18/1/14, 1060 Wien

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, 31. Jänner 2019

Johannes Maschler

Acknowledgements

First and foremost I want to thank my advisor Günther Raidl for giving me the opportunity to conduct my doctoral studies. I am grateful for the wonderful collaboration, fruitful suggestions, and for guidance leading me to this point. Not only I got the chance to teach and to help supervise master's theses, but I also was able to work on a meaningful real-world problem and interesting spin-offs. Moreover, he found a way to fund my position through all these years which was in my case not a trivial task.

Besides my advisor, I want to express my gratitude to Luca Di Gaspero and Nysret Musliu for evaluating this thesis.

I want to thank EBG MedAustron GmbH, Marie Curie-Straße 5, 2700 Wiener Neustadt, Austria, for their cooperation and for partially funding the project leading to this thesis.

I also like to thank my former and current colleagues at the Algorithms and Complexity Group for the friendly environment, the vivid discussions, and alternative viewpoints. To this end, I want to especially mention Martin Riedler and Matthias Horn with whom I coauthored articles and worked closely together.

Last but not least, I want to thank my family and my girlfriend Christina for their ongoing support and encouragement that provided me with the confidence to overcome all obstacles and made this work in the end possible.

Abstract

Particle therapy is an innovative and highly promising option to provide cancer treatments. In particle therapy protons or carbon ions are accelerated and redirected to one of several treatment rooms that all share the single accelerator. Each treatment starts with some specific preparations such as immobilization and positioning of the patient. Then the irradiation using the particle beam takes place. Afterwards, some additional imaging needs to be done before the patient can leave and the treatment room becomes available for a next patient. Ideally, one aims to find a schedule where the particle beam can be directly switched from one treatment room to next without significant breaks. Accomplishing this goal is far from trivial and requires elaborate scheduling techniques.

We study the midterm planning problem of such a particle therapy treatment center, which consists of scheduling therapies over the next few months. Therapies involve 8 to 35 *daily treatments* (DTs) that need to be provided on subsequent days. There are various constraints determining the succession of the therapies' DTs including allowed starting days and minimum and maximum numbers of days between subsequent DTs. Next to assigning DTs to days also detailed schedules for each of these days have to be provided. For their execution DTs require several resources for a specified duration. Among those resources are the treatment rooms that are typically required for the whole DT and the particle beam which is needed only during the irradiation. Resources are available for a regular and an extended service time. The objective is to find a schedule for all DTs that minimizes the use of extended service time as well as the therapies' finishing days.

Since initial investigations soon showed that solving this problem with exact techniques is clearly out of reach, we focus on heuristic and metaheuristic techniques. We propose a constructive heuristic that first assigns DTs therapy-wise to days and afterwards creates schedules for the individual days. Based on this constructive heuristic we provide an *iterated greedy* (IG) metaheuristic that is able to substantially improve upon the already reasonable results of our greedy heuristic. This initially simple metaheuristic is then improved further in several steps. We first revise the IG's main operators and incorporate a new powerful local search method. Afterwards, an advanced surrogate model is developed to further improve the assignment of DTs to days. In the last stage of extensions we consider the important additional aspect that the DTs belonging to the same therapy should be planned roughly at the same time. In the conducted computational study we show that each of these extensions results in a significant improvement of the approach.

Next we focus in more detail on the isolated scheduling of DTs at single days. In the considered more abstract problem we are given a set of jobs with similar characteristics as DTs. Due to time windows only a subset of the jobs can typically be feasibly scheduled. The objective is to maximize the total prize associated with each of the scheduled jobs. This prize-collecting aspect reflects the need of delaying less critical DTs to later days if for the considered day the facility's capacity is reached.

We study for this problem the application of *decision diagrams* (DDs). Relatively compact relaxed and restricted DDs are employed that represent discrete relaxations for the problem and encode heuristic solutions, respectively. The prize-collecting aspect of the problem at hand has not been studied in this form before and provides new interesting challenges. We investigate different methods for compiling relaxed and restricted DDs for our sequencing problem. We start by adapting the two proposed compilation approaches in the literature and are, to the best of our knowledge, the first who directly compare them experimentally. These traditional compilation methods have, however, the disadvantage that they can cause for our problem major redundancies in the DDs. For this reason, we propose a novel construction scheme for relaxed DDs inspired by A* search that provides more flexibility. In comparison to the two traditional approaches we can compile for our problem in shorter time substantially smaller DDs that yield stronger bounds. We show further how the information already contained in a previously compiled relaxed DD can be exploited during the construction of restricted DDs. With our novel approaches we are in many cases able to construct a higher quality relaxed DD and restricted DD in less than half of the time than compiling both with conventional approaches. The general approach applied here appears to be also promising for problems in other applications domains.

We give further an overview of developed solution approaches for two additional optimization problems. One is a strip packing variant that is solved using a logic-based Benders decomposition. The other is a resource-constrained project scheduling problem that requires scheduling in a high temporal resolution. A so-called time-bucket relaxation of the problem is iteratively refined in the suggested solution approach.

Kurzfassung

Partikeltherapie ist eine innovative und höchst vielversprechende Option zur Krebsbehandlung. In der Partikeltherapie werden Protonen und Kohlenstoffionen beschleunigt und in einen von mehreren Behandlungsräumen geleitet, die sich den Teilchenbeschleuniger teilen. Jede Behandlung startet mit mehreren spezifischen Vorbereitungen, wie die Immobilisierung und die Positionierung des Patienten. Danach findet die Bestrahlung mithilfe des Partikelstrahles statt. Anschließend werden noch zusätzliche Bildgebungsverfahren durchgeführt bevor der Patient den Behandlungsraum verlässt und dieser für den nächsten Patienten bereit ist. Idealerweise wird ein Zeitplan gesucht bei welchen der Partikelstrahl direkt von einem Behandlungsraum zum nächsten geleitet werden kann ohne das signifikante Stehzeiten entstehen. Dieses Ziel zu erreichen ist nicht trivial und erfordert wohldurchdachte Planungstechniken.

Wir untersuchen das mittelfristige Planungsproblem von solch einem Zentrum für Partikeltherapie, welches aus dem Planen von Therapien für die nächsten Monate besteht. Therapien umfassen 8 bis 35 Behandlungen (*Daily Treatments* (DTs)), die an aufeinander folgenden Tagen stattfinden müssen. Es gibt verschiedene Einschränkungen, welche die Aufeinanderfolge der DTs von Therapien bestimmen. Zu diesen gehören erlaubte Starttage und Minimal- und Maximalanzahlen von Tagen zwischen zwei DTs. Neben der Zuweisung von DTs zu Tagen muss auch ein detaillierter Zeitplan für jeden dieser Tage gefunden werden. Für die Ausführung eines DTs werden mehrere Ressourcen für vorgegebene Zeitspannen benötigt. Zu diesen Ressourcen gehören die Behandlungsräume, die üblicherweise während des gesamten DTs benötigt werden, und der Partikelstrahl, welcher nur für die eigentliche Bestrahlung verwendet wird. Ressourcen sind während regulärer und erweiterter Dienstzeiten verfügbar. Das Ziel ist es einen Zeitplan für alle DTs zu finden, der die verwendete erweiterte Dienstzeit und das Ende aller Therapien minimiert.

Unsere ersten Untersuchungen zeigten, dass das exakte Lösen dieses Problems in akzeptabler Laufzeit in der Praxis nicht möglich ist. Wir setzen deshalb auf heuristische und metaheuristische Techniken. Wir präsentieren eine konstruktive Heuristik, die zuerst DTs therapieweise den Tagen zuweist und anschließend Zeitpläne für die einzelnen Tage erstellt. Aufbauend auf dieser Konstruktionsheuristik erstellen wir eine erste *Iterated Greedy* (IG) Metaheuristik, welcher es möglich ist die bereits plausiblen Lösungen der Konstruktionsheuristik erheblich zu verbessern. Diese anfänglich einfache Metaheuristik

wird in mehreren Schritten verbessert. Zuerst überarbeiten wir die zentralen Bestandteile des IGs und führen eine neue leistungsstarke lokale Suchmethode ein. Anschließend wird ein vertiefendes Ersatzmodell entwickelt, um die Zuteilung von DTs auf Tage zu verbessern. In der letzten Ausbaustufe betrachten wir einen wichtigen zusätzlichen Aspekt bei dem DTs derselben Therapie zu ähnlichen Zeiten stattfinden sollen. In den durchgeführten Experimenten zeigte sich, dass jede dieser Erweiterungen zu einer signifikant Leistungssteigerung des Ansatzes führt.

In weiterer Folge konzentrieren wir uns konkreter auf das isolierte planen von DTs an einzelnen Tagen. In diesem allgemeiner gehaltenen Problem ist eine Menge von Aufträgen mit Charakteristiken ähnlich zu jenen der DTs gegeben. Aufgrund von Zeitfenstern ist es im Allgemeinen nur möglich eine Teilmenge der Aufträge zu planen. Das Optimierungsziel ist es die Summe der Preise, die den geplanten Aufträgen zugeteilt sind, zu maximieren. Dieser preissammelnde Optimierungsaspekt reflektiert die Notwendigkeit weniger kritische DTs auf spätere Tage zu verschieben, falls an einem betrachtenden Tag die Kapazität des Zentrums bereits erreicht ist.

Wir untersuchen für dieses Problem den Einsatz von Entscheidungsdiagrammen (*Decision Diagrams* (DDs)). Dabei werden relativ kompakte relaxierte und eingeschränkte DDs eingesetzt. Diese stellen diskrete Relaxierungen für das Problem dar bzw. kodieren heuristische Lösungen. Der preissammelnde Optimierungsaspekt unseres Problems wurde in dieser Form noch nicht betrachtet und bietet neue interessante Herausforderungen. Wir entwickeln mehrere Methoden zur Erstellung von relaxierten und eingeschränkten DDs für unser Sequenzproblem. Zuerst adaptieren wir die zwei in der Literatur vorgeschlagenen Ansätze und sind, nach bestem Wissen, die Ersten die diese direkt experimentell vergleichen. Diese herkömmlichen Erstellungsmethoden von DDs haben bei unserem Problem den Nachteil, dass diese nicht zu vernachlässigende Redundanzen in den DDs erzeugen können. Aus diesem Grund schlagen wir eine neuartige, von der A* Suche inspirierte, Konstruktionsmethodik vor, die mehr Flexibilität bietet. Im Vergleich zu den herkömmlichen Ansätzen kann diese Methode für unser Problem in kürzerer Zeit wesentlich kleinere DDs erzeugen, die zudem stärkere Schranken liefern. Zusätzlich zeigen wir wie die Information, welche bereits in zuvor erstellten relaxierten DDs steckt, in der Konstruktion von eingeschränkten DDs ausgenutzt werden kann. Mit unseren neuen Ansätzen können wir in vielen Fällen qualitativ höhere relaxierte und eingeschränkte DDs in weniger als der Hälfte der Zeit erstellen, die für die Erzeugung beider DDs mit den herkömmlichen Methoden benötigt wird.

Weiterführend geben wir einen Überblick über Lösungsmethoden für zwei zusätzliche Optimierungsprobleme. Eines ist eine Variante des Streifenpackproblems, die mithilfe einer Logik-basierten Benders-Zerlegung gelöst wird. Das andere ist ein Terminplanungsproblem, das eine besonders feine Zeitauflösung benötigt. Eine Zeitintervallrelaxierung für das Problem wird in dem vorgeschlagenen Lösungsansatz iterativ verfeinert.

Contents

CHAPTER 1

Introduction

The number of new cancer cases in 2012 amounted to about 14.1 million worldwide (not including skin cancer other than melanoma). In the same year cancer caused about 14.6% of all human deaths [109], making it one of the ten most common causes of death [115]. Furthermore, these numbers are increasing rapidly. Already in 2018 estimations reckon to be 18.1 million new cancer cases and 9.6 million cancer deaths [17]. According to projections the number of new cancer cases will reach 22.2 million by 2030 [16]. Consequently, a serious amount of research and investments has been contributed to provide and develop treatments. A widely applied treatment option is radiotherapy. Usually *linear accelerators* (LINACs) are used to provide treatment in conventional external beam therapy (electron or photon therapy). More recently the option to treat tumors with particle beams (like protons or carbon ions) has shown to be promising. Compared to conventional radiotherapy, this modern approach allows to reduce the radiation exposure to adjacent healthy tissue substantially. Nevertheless, also the investment costs to build such centers are considerably higher than for traditional centers. It is therefore particularly important to utilize available resources in particle therapy centers as efficiently as possible in order to maximize the benefits patients receive over time.

In typical photon and electron radiotherapy it is common that a single LINAC serves a dedicated room exclusively. In contrast to LINACs, particle beams are produced by either cyclotrons or synchrotrons which serve two to five treatment rooms alternatingly. We consider here more specifically the radiotherapy treatment center MedAustron[1] located in Wiener Neustadt, Austria. This emerging facility is currently one of the most modern of its kind. In Wiener Neustadt proton and carbon ion beams are being produced by a synchrotron serving three treatment rooms, each with different capabilities.

The treatment rooms are occupied for some time before and after an actual irradiation of a patient. These surrounding tasks involve the immobilization and positioning of the

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

patient, additional imaging and the time required for the patient to leave the treatment room. To avoid undesirable idle times on the beam as far as possible the treatment rooms are used in parallel. As the duration of the tasks executed in the context of the performed irradiations vary, a carefully arranged schedule has to be found. Moreover, several side-constraints and resource dependencies need to be respected. Accomplishing this goal is far from trivial and requires elaborate scheduling techniques.

In the first part of this thesis we consider the midterm planning problem of such a particle therapy treatment center, in which an effective plan has to be found for performing a larger number of therapies over the next few months. Therapies will typically involve a sequence of 8 to 35 *daily treatments* (DTs) provided on more or less subsequent days. One DT corresponds to the above mentioned tasks surrounding and including an irradiation of a patient. There are various constraints that determine possible starting days and the frequency for the therapies' DTs. Among others, the therapies' first DT has to be performed either on Mondays or Tuesdays and after that there should be provided typically between four and five DTs per week. Next to assigning DTs to days also detailed starting times on these days have to be determined in order to obtain a valid solution. DTs require several resources during specified parts of their execution time. Typically, one dedicated treatment room is needed for the whole duration of the DT, while the beam is only necessary for the part that represents the irradiation. Other demanded resources include patients, radio oncologists and an anesthetist. Moreover, resources have restrictions in their availability. We consider regular and extended service time windows, where the use of the latter produces additional cost. Furthermore, the resources' service time windows can be interrupted by unavailability periods. Our primary goal is to minimize the use of extended service time, while completing therapies as early as possible.

We start with formalizing this midterm planning problem in terms of a *mixed integer linear program* (MILP) model. Solving this model for problem instances of relevant size is, however, in practice not possible. Therefore, we focus on heuristic and metaheuristic techniques. We propose a therapy-wise construction heuristic, which first assigns all DTs to days and then determines schedules for each day individually. Based on this constructive heuristic we provide a first *iterated greedy* (IG) metaheuristic [107] that clearly outperforms a *greedy randomized adaptive search procedure* (GRASP) [102] in our experiments. We further identify several limitations in this first IG preventing the approach of exhausting its full potential. In a first step we revise the IG's destruction and construction operators and replace the so far employed local improvement operator by a more powerful local search method. Afterwards, we reconsider the first step of our therapy-wise construction heuristic that assigns all DTs to days. The decisions made during this step are based on an estimation whether the assigned DTs will result in a schedule that uses extended service time. So far we employed a simple lower bound which is replaced by an estimation based on a more advanced surrogate model. Our experiments indicate that both of these modifications of our initial approach yield substantially better results. We continue our efforts by considering an extended problem variant that covers

the additional aspect that starting time variations of DTs belonging to the same therapy should not exceed specified thresholds. A particular challenge of this new aspect is that the scheduling of the DTs on each day is not independent anymore. We revise our IG metaheuristic and adapt its components to the extended problem scenario. In the computational study we assess the quality of each of our IG's components by comparing them to a reference approach. The metaheuristic is advantageous on all considered benchmark instances.

In the second part of this thesis we study a problem which is motivated by the above real-world problem but focuses on the scheduling of DTs within days. More specifically, we are given a set of jobs (DTs) from which a subset needs to be selected and feasibly scheduled. Each job requires a common resource which is shared among all jobs and one of multiple secondary resources. While the secondary resource is occupied for the job's whole processing time, the common resource is used only for a specified part of the job's execution. Moreover, jobs can be performed only within one of their given time windows. Due to these time windows, it is in general not possible to feasibly schedule all jobs. Moreover, each job has an associated prize and our objective is to find a feasible schedule for a subset of jobs that maximizes the total prize. The common resource naturally corresponds to the particle beam and the secondary resources correlate with the treatment rooms in our particle therapy scenario. The prize-collecting aspect is motivated in our real-world setting by the fact that we consider only a single day of limited availability and it is in general not possible to perform all available DTs. Consequently, we have to find a schedule for a best subset of DTs. Unscheduled DTs have to be performed at other days. The job prizes may reflect the urgencies of the respective DTs, or may correspond to the duration of the treatments.

This job sequencing problem is approached with *decision diagrams* (DDs) [12]. Essentially, DDs are rooted directed acyclic multigraphs that compactly represent a problem's set of feasible solutions. In about the last decade, DDs have been recognized to be powerful tools for certain optimization problems. Relaxed DDs and restricted DDs are typically employed to cope with the often exponential number of solutions. While the former represent discrete relaxations of the problem and provide dual bounds, the latter encode subsets of the solutions and allow deriving heuristic solutions and primal bounds. We investigate different methods for creating relaxed and restricted DDs for our job sequencing problem. We start by adapting the two main approaches that have been proposed in the literature for compiling DDs. The first method starts with an empty DD and constructs the DD layer by layer. The second approach takes a simplistic DD as input and iteratively performs refinements. To the best of our knowledge, we are the first who directly compare the two techniques experimentally. The results indicate that depending on the instance class the second method produces DDs either with similar or stronger bounds. These traditional compilation methods are strongly layer oriented, which may, however, cause major redundancies in the DDs for our problem. Therefore, we propose a novel construction scheme for relaxed DDs inspired by A* search that avoids explicit layers and, thus, provides more flexibility. Our experiments show that compared to the

traditional approaches DDs can be build faster, they are substantially smaller, and they provide stronger bounds. Finally, we show how to exploit the information contained in a previously compiled relaxed DD during the construction of restricted DDs. With our last two approaches we are often able to construct a relaxed DD and a restricted DD in less time than the traditional construction of restricted DDs. In doing so, we are not only able to derive frequently stronger primal bounds, but also provide dual bounds which represent quality guaranties.

Another considered problem is a resource-constrained project scheduling problem with a high temporal resolution. The problem is again inspired by scheduling DTs within days and consists of scheduling a set of activities. There are precedence constraints that enforce minimum and maximum time lags between activities, which require a set of resources during their execution. The objective is to minimize the completion time of the last activity. An important aspect of the problem at hand is the assumption that a fine-grained time resolution is required. This is motivated by the fact that in particle therapy the irradiation times are precisely known and idle times on the beam resource should be avoid as far as possible. In this thesis we give only an overview over the solution method that is based on aggregating time to obtain a relaxation. The proposed approach iteratively solves and refines the relaxation and aims at deriving strong heuristic solutions together with dual bounds. The approach, however, is complete in the sense that it is also guaranteed to converge to an optimal solution if enough time is given.

Finally, we also address a strip packing problem in which rectangles have to be placed onto a strip of fixed width and unlimited height. The objective is to minimize the total used height. We consider further the restriction that all rectangles have to be obtainable by three stages of so-called guillotine cuts. Strip packing is not directly related with patient scheduling in particle therapy, but there is still a strong relationship between strip packing and certain scheduling problems. For the considered strip packing variant a *logic-based Benders decomposition* (LBBD) approach is developed that outperforms a corresponding compact MILP model.

1.1 Structure of the Thesis

We start in the following chapter with an overview of the most central concepts and techniques applied within this thesis. To this end, we first define optimization problems in a general way and give an overview of the solution methods that are most relevant w.r.t. this thesis.

Chapter 3 is dedicated to the midterm planning part of the particle therapy treatment center MedAustron. After giving the specifics of the treatment center and the scheduling problem at hand, we review related literature. Then, we give a problem definition and specify a MILP model for the basic and extended problem variant. To obtain first solutions we present next a constructive heuristic that also serves as basis for an IG and a GRASP. The IG is developed further. First we exchange components of the IG in order to preserve more information between iterations. Afterwards, we describe a surrogate

model to improve the assignment of DTs to days by the constructive heuristic. Later we consider the extended problem variant and extend the IG accordingly. Finally, we describe the generated benchmark instances and explain the results of the conducted experiments.

Chapter 4 is devoted to the prize-collecting job sequencing problem. We study for this problem the application of DDs. After revising related literature and giving a problem definition we provide a recursive model on which the DDs are based. We investigate in total four different compilation techniques for DDs. The first two are adaptions of conventional compilation methods proposed in the literature. The next one is conceptually new and provides more flexibility than the traditional methods. While this construction approach provides relaxed DDs and dual bounds, the last construction technique produces restricted DDs yielding heuristic solutions and primal bounds. In the computational study we consider instances inspired by our therapy scenario as well as another application in avionics and compare the proposed approaches.

In Chapter 5, the strip packing problem and the *resource-constrained project scheduling problem* (RCPSP) are considered. For the strip packing problem the LBBD approach with two kinds of Benders cuts is proposed and is experimentally compared with a compact formulation. A matheuristic is provided for the studied RCPSP variant. The matheuristic's main building block is the so-called time-bucket relaxation which is iteratively refined.

Finally, Chapter 6 concludes this thesis with a summary of the key findings and an outlook on future research directions.

CHAPTER 2

Methodology

This chapter gives an overview on the most central concepts and techniques used throughout this thesis. We start by defining in the next section the considered problems and briefly discuss why some of them are hard to solve in theory and practice. In the following three sections, we shift the focus on the methods to tackle these problems. Rather than giving a overview over the research area's most prominent approaches, our aim is to survey the most relevant techniques w.r.t. the presented works. We will consider in Section 2.2 exact methods from which we highlight branch-and-bound, dynamic programming, mixed integer linear programming, and its special case linear programming. Algorithms that can be classified as heuristic and metaheuristic procedures are discussed in Section 2.3, from which we will consider constructive heuristics, local searches, greedy randomized adaptive search procedures, iterated greedy algorithms, and variable neighborhood searches. In Section 2.4 the concept of applying decision diagrams for optimization is motivated. Moreover, we give an overview of the most promising approaches proposed in this research area.

2.1 Combinatorial Optimization Problems

The problems considered in the course of this thesis are defined as *combinatorial optimization problems* (COPs). Following the definitions from Papadimitriou and Steiglitz [92] a COP specifies a set of instances. An instance consists of a pair $(\mathscr{S}, f)$ of a finite set of feasible solutions $\mathscr{S}$ and an objective function $f\colon \mathscr{S} \to \mathbb{R}$ that maps a cost value to each of the solutions. We search for a solution $s^* \in \mathscr{S}$ with either minimal or maximal cost value, i.e., with either $f(s^*) \leq f(s)$ or $f(s^*) \geq f(s)$ for all $s \in \mathscr{S}$. Such a solution s^* is called *globally optimal* w.r.t. the given instance. Moreover, searching for a solution that minimizes objective function f is equivalent to looking for a solution that maximizes $-f$. Hence, we can reformulate each COP s.t. its objective function has to be maximized.

The structure of the solutions in $\mathscr{S}$ is typically problem dependent. They range from subsets or permutations of given elements to vectors of integral values. For most COPs it is, however, not practicable to explicitly state instances due to their enormous number of feasible solutions. Instead, problem instances are given implicitly by parameters that allow to precisely construct every solution in $\mathscr{S}$ and their corresponding cost value. An instance for the classic 0–1 knapsack problem is implicitly specified by a capacity b and a set I of items each having a weight w_i and a value v_i. The set of feasible solutions are all subsets of I with total sum of weights less than or equal to the capacity, i.e., $\mathscr{S} = \left\{ s \in 2^I \mid \sum_{i \in s} w_i \leq b \right\}$. The cost of a solution is the total value of the items in the subset, i.e., $f(s) = \sum_{i \in s} v_i$. The optimization goal is to find a feasible solution that maximizes the total value of the items, or formally solving $\max \{ \sum_{i \in s} v_i \mid s \in \mathscr{S} \}$.

For many relevant COPs finding globally optimal solutions for a given instance is difficult in practice. From the theoretical viewpoint this difficulty is often rooted in the NP-hardness of the considered problems. Under the common assumption that NP≠P, correctly solving an instance of an NP-hard problem requires in the worst case an exponential number of steps [37]. Approaches to tackle such COPs can be categorized into exact methods, approximation algorithms, and heuristics. Exact algorithms guarantee to find a globally optimal solution but may require exponential computation time. In practice, their applicability is strongly dependent on the specific COP and the size of the considered instances. In contrast, approximation algorithms terminate after a polynomially bounded computation time with solutions that are guaranteed to be close to optimal. Heuristic algorithms run also in polynomial time but focus on finding high quality solutions without guaranteeing their optimality.

2.2 Exact Methods

The main property of exact methods is that they guarantee to find optimal solutions. For COPs belonging to the complexity class P exact algorithms can be found that complete in polynomial time. In many cases optimal solutions can be directly constructed by exploiting problem properties. The COPs considered in this thesis are, however, NP-hard which entails that finding an optimal solution takes in the worst case exponentially many steps. Such COPs are typically approached with enumeration schemes. As a naive exhaustive enumeration is not viable in practice, the key idea is to exclude as many solutions from explicit consideration while still ensuring that at least one of the optimal solutions is found.

We start in the following sections with giving an overview of two basic enumeration schemes: branch-and-bound and dynamic programming. Afterwards, in Section 2.2.3 the more advanced mixed integer linear programming is discussed which has shown to be effective for many NP-hard COPs. Worth mentioning is also constraint programming that provides compared to mixed integer linear programming a rich set of constraint types. As constraint programming has not been applied in the presented work we omit an overview here and refer to the textbook by Rossi et al. [106].

2.2.1 Branch-and-Bound

A classic exact method for approaching NP-hard COPs is *branch-and-bound* (see e.g. [92]). We will discuss the technique here only on a high level, a more detailed example will be presented in the context of mixed integer linear programming in Section 2.2.3. The branch-and-bound method is based on recursively partitioning the considered COP's set of feasible solutions in two or more disjunct subsets. This process is called branching and results in a search tree where the root node represents the whole problem and each of the successor nodes is associated with a decreasingly smaller subproblem. Eventually, the subproblems become small enough to solve them trivially. Each found solution is a primal bound on the whole problem's optimal objective value. The second major component of a branch-and-bound approach is a bounding algorithm that provides a dual bound on a considered subproblem's optimal cost. Whenever such a dual bound is worse than the so far best primal bound of any subproblem, then the respective subproblem cannot contain a better solution. Consequently, we exclude the subset from further consideration and continue with the next one.

For the 0–1 knapsack problem, which has been introduced in the previous section, a suitable branching rule would partition a set of feasible solutions into a subset where an item is part of all solutions and another subset consisting of the remaining solutions. An example for a bounding algorithm is to pack first all items already fixed by the branching rule and then the remaining items in decreasing value/weight-ratio order. The first item that cannot be feasibly packed is assigned partially. The resulting total value is an upper bound on the subproblems objective value [66].

2.2.2 Dynamic Programming

Another classic principle is *dynamic programming* (DP) [5] that solves a recursive formulation of the problem and exploits recurring subproblems. The basic components of such a recursive formulation are states, transitions between states, and transition costs. The states are typically partitioned into $n+1$ subsets, called stages, where n is the number of decisions we need to make for obtaining a solution of the COP. The first stage consists of a single root state with no decisions yet made. A transition applies a decision at a certain cost on a given state and yields a successor state located in a subsequent stage. Sequences of feasible transitions starting with the root state ending at a so-called terminal state located in the last stage describe solutions and the total occurred cost corresponds to the objective value. For solving a considered COP, we are interested in a transition sequence from the root state to a terminal state that with optimal cost. An essential observation is that in the corresponding transition sequence all intermediate states are reached by optimal cost as well. Consequently, we are not interested in all feasible transition sequences starting from the root state, but only in those that reach a state with optimal cost. Solving a DP formulation typically entails storing the optimal cost for all possible states stage-by-stage in a table. The optimal solution value can then be found in the last stage. For a more comprehensive introduction into DP we refer to the textbooks [12, 92, 69].

To showcase DP we consider again the 0–1 knapsack problem that has been introduced in Section 2.1. We follow here loosely the corresponding example in [12]. Let $\{1, \dots, n\}$ be the set of n items in I. The decision to be made in each transition from stage i to stage $i+1$ is whether the item $i \in I$ is packed or not. The states represent the total weight of all packed items. In particular, the root state has a total weight of 0. There are two options for state transitions: one where the item corresponding to the stage is packed and one where the item is skipped. The resulting state in the former case is obtained by adding the weight of the packed item to the original state, while in the latter case the state does not change. If a transition exceeds the available capacity b, then the special infeasible state $\hat{0}$ is obtained. Any further transition from $\hat{0}$ yields $\hat{0}$ again. The state transitions are formally defined by a function that takes as input a state s and a decision $x_i \in \{0, 1\}$ whether item i is packed, i.e.,

$$\tau(s, x_i) = \begin{cases} s + w_i x_i & \text{if } s + w_i x_i \leq b \\ \hat{0} & \text{else.} \end{cases} \tag{2.1}$$

The transition has a cost of v_i when item i is packed and 0 otherwise. The 0–1 knapsack problem can now be formulated by the recursion

$$P_i(s) = \max \left\{ 0, v_i x_i + P_{i+1}(\tau(s, x_i)) \mid i \leq n,\ x_i \in \{0, 1\},\ \tau(s, x_i) \neq \hat{0} \right\}, \tag{2.2}$$

where calculating $P_1(0)$ solves the considered problem instance.

2.2.3 Mixed Integer Linear Programming

Mixed integer linear programming is one of the most prominent techniques to solve COPs exactly. In this section mixed integer linear programming and its most important special cases are defined, and we briefly discuss how they can be solved. We follow here closely the first and later the eighth chapter of the textbook by Conforti et al. [27].

A *mixed integer linear program* (MILP) [27, p.2] is of the form

$$\begin{aligned} \max\ & \mathbf{c}^T \mathbf{x} + \mathbf{h}^T \mathbf{y} \\ \text{subject to}\ & \mathbf{A}\mathbf{x} + \mathbf{G}\mathbf{y} \leq \mathbf{b} \\ & \mathbf{x} \in \mathbb{Z}^n_{\geq 0} \\ & \mathbf{y} \in \mathbb{R}^p_{\geq 0}, \end{aligned} \tag{2.3}$$

where the vectors $\mathbf{c} \in \mathbb{R}^n$, $\mathbf{h} \in \mathbb{R}^p$, $\mathbf{b} \in \mathbb{R}^m$ and the matrices $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{G} \in \mathbb{R}^{m \times p}$ are the given input, and we seek to optimize the variables in $\mathbf{x}$ and $\mathbf{y}$. An MILP describes a COP's instance $(\mathscr{S}, f)$ by its set of feasible solutions

$$\mathscr{S} = \left\{ (\mathbf{x}, \mathbf{y}) \in \mathbb{Z}^n_{\geq 0} \times \mathbb{R}^p_{\geq 0} \mid \mathbf{A}\mathbf{x} + \mathbf{G}\mathbf{y} \leq \mathbf{b} \right\} \tag{2.4}$$

and the objective function $f = \mathbf{c}^T \mathbf{x} + \mathbf{h}^T \mathbf{y}$.

Instances for the 0–1 knapsack problem, which we already discussed in Section 2.1, can be naturally modeled by the following MILP

$$\max \sum_{i=1}^{|I|} v_i x_i \tag{2.5}$$

$$\text{subject to } \sum_{i=1}^{|I|} w_i x_i \leq b \tag{2.6}$$

$$\mathbf{x} \in \{0,1\}^{|I|}, \tag{2.7}$$

where we search for an assignment for the binary variables in $\mathbf{x}$ that satisfies the constraint (2.6) and maximizes the objective function (2.5).

There are two important special cases of MILPs. The first one are *integer linear programs* (ILPs) that have only integer variables, i.e., $p = 0$. The above model for the 0–1 knapsack problem is an example for an ILP, but since its variables can assume only the values 0 and 1 it is typically further categorized as a *0–1 linear program*. The fact that the 0–1 knapsack problem is NP-hard [65, 92] allows us to conclude that ILPs are powerful enough to express NP-hard problems. In fact, deciding whether an MILP's set of feasible solutions $\mathscr{S}$ is nonempty is NP-complete [28].

The other special case are *linear programs* (LPs) where all variables are assumed to be rational, i.e., $n = 0$. In contrast to ILPs, LPs can be solved in polynomial time. The first algorithm with polynomial-time worst-case complexity was the ellipsoid method [67], which is, however, only of theoretical interest. Nowadays, LPs are typically solved with the simplex method [30] or the interior point algorithm [64]. Both algorithms provide an excellent performance in practice, however, only the interior point method is a polynomial-time algorithm.

Although these algorithms cannot be directly applied for the more general case where integer variables are present, they can be used to solve powerful relaxations of MILPs. Given an MILP of the form (2.3) we obtain a *linear programming relaxation* by allowing the variables $\mathbf{x}$ to assume nonnegative rational values instead of just nonnegative integer values. The feasible solutions of the resulting LP are given by the set

$$P_0 = \left\{(\mathbf{x}, \mathbf{y}) \in \mathbb{R}^n_{\geq 0} \times \mathbb{R}^p_{\geq 0} \mid \mathbf{A}\mathbf{x} + \mathbf{G}\mathbf{y} \leq \mathbf{b}\right\}, \tag{2.8}$$

where, clearly, every feasible solution for the MILP is also a feasible solution for its linear programming relaxation, i.e., $\mathscr{S} \subseteq P_0$. Moreover, the objective value of an optimal solution to the linear programming relaxation is an upper bound on the optimal objective value of the MILP. In contrast to the original MILP, its linear programming relaxation is an LP which in general can be solved much faster. Furthermore, if it turns out that all $\mathbf{x}$ variables in an optimal solution of the linear programming relaxation are integral, then this solution is also a feasible and optimal solution for the MILP.

The linear programming relaxation of an MILP cannot only be used to obtain an upper bound, but it can serve as a starting point for a branch-and-bound method that guarantees

to find optimal solutions. The central idea is to generate and solve a sequence of LPs that represents an increasingly tighter approximation of the MILP's set of feasible solutions. We repeat until the optimal solution of the relaxation is contained in $\mathscr{S}$ and, thus, we have found an optimal solution for the MILP. Basically, the refinement of the MILP's linear programming relaxation is performed as follows. Suppose that in an optimal solution of the linear programming relaxation $(\mathbf{x}^0, \mathbf{y}^0) \in P_0$ value x_j^0, where $1 \leq j \leq n$, is fractional. As the corresponding variable in MILP has to be integer, either $x_j \leq \lfloor x_j^i \rfloor$ or $x_j \geq \lceil x_j^i \rceil$ has to hold. Hence, the union of the sets

$$P_1 = \left\{(\mathbf{x}, \mathbf{y}) \in P_0 \mid x_j \leq \lfloor x_j^i \rfloor\right\}, \qquad P_2 = \left\{(\mathbf{x}, \mathbf{y}) \in P_0 \mid x_j \geq \lceil x_j^i \rceil\right\}, \tag{2.9}$$

is still a valid relaxation of $\mathscr{S}$ in that $\mathscr{S} \subseteq P_1 \cup P_2$. If we solve the two LPs

$$\max\left\{\mathbf{c}^T\mathbf{x} + \mathbf{h}^T\mathbf{y} \mid (\mathbf{x}, \mathbf{y}) \in P_1\right\}, \qquad \max\left\{\mathbf{c}^T\mathbf{x} + \mathbf{h}^T\mathbf{y} \mid (\mathbf{x}, \mathbf{y}) \in P_2\right\}, \tag{2.10}$$

then the best of both optimal objective values is a possibly tighter upper bound for the MILP's optimal solution value.

Algorithm 2.1: LP-based branch-and-bound [27, p.10]

Input: a MILP of the form $\max\{\mathbf{c}^T\mathbf{x} + \mathbf{h}^T\mathbf{y} \mid (\mathbf{x}, \mathbf{y}) \in \mathscr{S}\}$

$P_0 \leftarrow$ linear programming relaxation of $\mathscr{S}$;
$L \leftarrow \{P_0\}$;
$(\mathbf{x}^*, \mathbf{y}^*) \leftarrow \emptyset$;
$z^{\text{lb}} \leftarrow -\infty$;
while $L \neq \emptyset$ **do**
  choose P_i from L and remove it;
  solve $\max\{\mathbf{c}^T\mathbf{x} + \mathbf{h}^T\mathbf{y} \mid (\mathbf{x}, \mathbf{y}) \in P_i\}$;
  if $P_i = \emptyset$ **then**
    continue with next element in L;
  let $(\mathbf{x}^i, \mathbf{y}^i) \in P_i$ be an optimal solution and z_i is objective value;
  if $z_i \leq z^{\text{lb}}$ **then**
    continue with next element in L;
  if $\mathbf{x}^i \in \mathbb{Z}^n_{\geq 0}$ **then**
    $z^{\text{lb}} \leftarrow z_i$;
    $(\mathbf{x}^*, \mathbf{y}^*) \leftarrow (\mathbf{x}^i, \mathbf{y}^i)$;
    continue with next element in L;
  choose an $x_j^i \in \mathbf{x}^i$ that is still fractional;
  $L \leftarrow L \cup \{\{(\mathbf{x}, \mathbf{y}) \in P_i \mid x_j \leq \lfloor x_j^i \rfloor\}, \{(\mathbf{x}, \mathbf{y}) \in P_i \mid x_j \geq \lceil x_j^i \rceil\}\}$;

The branch-and-bound algorithm stated in Algorithm 2.1 uses this principle repeatedly to find an optimal solution for the MILP given as input. To exclude the consideration of some special cases, we assume here that the MILP's optimal objective value is finite.

The method starts by adding to list L the feasible solutions set of the MILP's the linear programming relaxation. Throughout the algorithm this list L contains the parts of this initial relaxation P_0 that still might contain the optimal solution of the given MILP. The so far best found feasible MILP solution and its objective value is stored in $(\mathbf{x}^*, \mathbf{y}^*)$ and z^{lb}, respectively.

In each iteration of the main loop a set of feasible LP solutions P_i is selected and removed from L. Then, the corresponding LP is solved. If the problem is infeasible, i.e., $P_i = \emptyset$, we can continue with the next element in L since P_i clearly does not contain a feasible solution w.r.t. the given MILP. If on the other hand the considered LP admits solutions, then we denote with $(\mathbf{x}^i, \mathbf{y}^i)$ the obtained optimal solution and the respective optimal solution value z_i. We can prune the considered LP and continue with the next LP if $z_i \leq z^{\text{lb}}$ because all contained MILP solutions in $P_i \cap \mathscr{S}$ have to be worse than current incumbent solution $(\mathbf{x}^*, \mathbf{y}^*)$. In case that $z_i > z^{\text{lb}}$ and that the values for all variables in $\mathbf{x}$ are integer, which means that $(\mathbf{x}^i, \mathbf{y}^i) \in \mathscr{S}$, we have found a new incumbent solution. Again, we do not have to consider the set P_i any further because we have already an optimal MILP solution for it and continue with the next iteration of the algorithm. Finally, if none of the above pruning conditions trigger the set P_i might still contain an optimal solution. Therefore, we select a variable x_j^i that is still fractional in our optimal solution of the considered LP. As described above, we create two new LPs where one has the additional constraint that $x_j \leq \lfloor x_j^i \rfloor$ and the other enforces that $x_j \geq \lceil x_j^i \rceil$. The corresponding sets of feasible solutions are then added to list L and we continue with the next iteration. The algorithm terminates if L becomes empty and, thus, the solution stored in $(\mathbf{x}^*, \mathbf{y}^*)$ has to be optimal.

Although modern MILP solver are nowadays very sophisticated, they frequently reach their limits for large and complex problem instances. Sometimes, when the problem exhibits a special structure, decomposition techniques might help to still successfully solve the problem at hand. Instead of solving a straightforward model, a sequence of smaller problems are solved and their results are combined appropriately. There are three classical decomposition techniques in the context of MILP. If from the constraints a subset of complicating constraints can be identified, then a *Lagrangian decomposition* [43] might be applicable. It is based on the so-called Lagrangian relaxation [36] that replaces these complicating constraints by terms in the objective function that penalize their noncompliance. The dual bounds that can be obtained from the Lagrangian relaxation are often stronger than those from classic LP relaxations which is for example of interest in branch-and-bound methods. The Lagrangian relaxation's quality is dependent by external parameters called the Lagrangian multipliers. Finding suitable values for the Lagrangian multipliers is a problem on its own and is often solved with iterative procedures such as the subgradient algorithm. Another approach is based on the *Dantzig-Wolfe reformulation* [31] of MILPs that usually requires a special block diagonal structure in the constraint matrix and yields a large number of variables. It is typically solved with the column generation method that considers a model restricted to a manageable subset of the variables. Optimal solutions of this model allow either to conclude that the

solution corresponds to an optimal one for the original problem or allows finding variables that may result in an improvement. Last but not least, the *Benders decomposition* [6] might be appropriate if a subset of complicating variables can be identified. The basic idea is to iteratively solve a relaxation of the problem that excludes these complicating variables, called the master problem, and induced Benders subproblems. These Benders subproblems allow deriving strengthening inequalities for the master problem. The Benders decomposition can be seen as a dual approach to the column generation because instead of adding variables to the model (columns in the constraint matrix) constraints are appended to the master problem (rows in the constraint matrix). We apply in Chapter 5 the *logic-based Benders decomposition* (LBBD) [52], a generalization of the Benders Decomposition that allows in particular the use of integral variables in the subproblems.

2.3 Heuristic and Metaheuristic Methods

Heuristic methods aim at providing good solutions in short time. In contrast to exact methods, the found solutions might not be optimal and often it is not possible to give any performance guarantee. They are applied either for supplying an exact method with a promising starting point or are the next best thing when the practical difficulty of the considered COP's problem instances prevents a successful application of exact methods.

We start with constructive heuristics in Section 2.3.1 which assemble a solution in general in polynomial many steps w.r.t. the instance size. Afterwards in Section 2.3.2, we describe local search, a technique that tries to improve a given solution. In addition, we discuss the concept of locally optimal solutions which are in principle solutions that local search cannot improve any further. In general, these locally optimal solutions are not globally optimal which motivated the development of *metaheuristics*.

Essentially, metaheuristics are problem independent frameworks to tackle difficult COP instances for which exact algorithms are not applicable. Metaheuristics cover a wide range of ideas how to efficiently search in a COP's set of feasible solutions for an optimal or near-optimal solution. All metaheuristics have in common that they provide a mechanism for *intensification* and *diversification*. Intensification is the exploration of similar solutions to a considered solution with the aim to find improving ones and is typically achieved with local search. Diversification is applied when the improvement possibilities of the intensification are exhausted and intends to move the focus of the metaheuristic to a new promising solution.

In the last 50 years a large variety of different metaheuristics have been proposed (for an overview see [39]). In the course of this chapter we will discuss in Section 2.3.3 the greedy randomized adaptive search procedure followed by the iterated greedy algorithm in Section 2.3.4. Both of them use a constructive heuristic as diversification mechanism and local search for intensification. In Section 2.3.5 a general variant of the variable neighborhood search is explained. This metaheuristic is primarily based on local search

and uses an alteration of it for diversification. The way these approaches are presented is inspired by the textbook from Blum and Raidl [15].

Further prominent metaheuristics that we will refer to are *genetic algorithms* (GAs) [101], *ant colony optimization* [33], *simulated annealing* [89], and *tabu search* [38]. GAs are motivated by evolution and maintain a set of solutions called population. In each iteration of the algorithm some solutions of the population are selected, recombined and possibly mutated to hopefully obtain better solutions for the next iteration. Ant colony optimization mimics the behavior of ants for finding shortest paths. Simulated annealing and tabu search are both based on local search. While the former allows deteriorations during local search with continually shrinking probability, the latter uses the search history for diversification.

2.3.1 Constructive Heuristics

Constructive heuristics have an instance of a COPs as input and aim at obtaining a solution in typically very short time. To this end, a constructive heuristic starts with an empty partial solution that is iteratively extended until we either have obtained a feasible solution or no feasible extension is possible.

Algorithm 2.2: Constructive Heuristic

Input: an instance of a COP
1 $s^P \leftarrow ()$;
2 **while** extension$(s^P) \neq \emptyset$ **do**
3 select element e from extension(s^P);
4 extend s^P by e;

Algorithm 2.2 shows the general concept of constructive heuristics in more detail. The solution to be built is denoted by s^P and is initially empty. We denote, with reference to the COPs considered in this work, partial solutions by sequences. In the main loop the partial solution is extended until we either obtain a complete solution or fail with an incomplete solution that cannot be feasibly extended. The latter case is for many problems excluded or highly unlikely. For a given partial solution s^P the set extension(s^P) contains all elements with which the partial solution can be feasibly extended. In case of the 0–1 knapsack problem extension(s^P) consists of all items that still fit within the capacity. In each step of the constructive heuristic we first select an element from the possible extensions and then extend the partial solution with it. The strategy for the selection of the next element has a high impact on the performance of the heuristic. A typical policy is to select the element that is most advantageous. Constructive heuristics that follow this strategy are called greedy heuristics. For the 0–1 knapsack problem we would choose for example an item having the largest value/weight ratio. Another option is to select the next element randomized to obtain more diverse solutions. Finally, the straightforward extension of the partial solution consists of appending the element at

the end of the sequence. Alternatively, more advanced techniques consider inserting the element within the sequence (e.g., the insertion heuristic by Nawaz et al. [87]).

2.3.2 Local Search and Variable Neighborhood Descent

Local search is a technique that aims at producing a series of improving solutions starting from a given solution. We begin with introducing the concepts of neighborhoods and local optimal solutions on which local search relies. As before, we follow the book by Papadimitriou and Steiglitz [92].

Given a solution s of a considered COP instance $(\mathscr{S}, f)$, a *neighborhood* $\mathscr{N}(s)$ defines a subset of feasible solutions from $\mathscr{S}$ that are close to s by some measure. Formally, a neighborhood is a function $\mathscr{N} : \mathscr{S} \to 2^{\mathscr{S}}$ that maps the feasible solutions of every given COP instance $(\mathscr{S}, f)$ to subsets of the feasible solutions. A natural neighborhood for the 0–1 knapsack problem is, for example, the k-change neighborhood that consists of all feasible solutions that differ from the considered solution s by at most k items.

A solution s of an instance $(\mathscr{S}, f)$ is *locally optimal* w.r.t. a neighborhood $\mathscr{N}$ if the solution's cost is better than or equal to the cost of all solutions in $\mathscr{N}(s)$, i.e., depending on whether we are minimizing or maximizing $f(s) \leq f(s')$ or $f(s) \geq f(s')$ for all $s' \in \mathscr{N}(s)$, respectively. Clearly, a globally optimal solution has to be also locally optimal w.r.t. any neighborhood, while the other way around does not necessarily hold.

Algorithm 2.3: Local Search

Input: solution s of a COP instance, neighborhood $\mathscr{N}$

```
1 while ∃s' ∈ N(s) with f(s') > f(s) do
2     s ← choose s' ∈ N(s) with f(s') > f(s);
```

Local search for COPs where the objective function has to be maximized is shown in Algorithm 2.3. The method assumes a solution and a neighborhood as input. In each step of the local search, which is typically called a *move*, the incumbent solution is replaced by a solution from its neighborhood with a better cost. The algorithm terminates if the current solution is locally optimal w.r.t. the neighborhood.

There are two basic rules that determine which solution is selected for replacing the incumbent solution if the neighborhood contains more than one solution with better cost: the *first improvement* and the *best improvement* rule. During the search for an improving solution we have to generate the solutions in the neighborhood and evaluate their costs. If we follow the first improvement rule, we stop creating solutions as soon as we have found a solution that has better costs than the incumbent solution and replace it. In contrast, the best solution of the neighborhood is used when performing best improvement. While the best solution might have better costs than the first obtained solution, finding the best solution requires in general to enumerate all solutions of the neighborhood. Hence, performing a best improvement step requires in expectation more

computation time. Nevertheless, following the best improvement rule typically yields a local optimal solution in fewer iterations of the local search.

A natural extension of the local search, as presented in Algorithm 2.3, is to consider more than one neighborhood. This is especially motivated by the fact that a locally optimal solution w.r.t. some neighborhood might still be improved by solutions from another neighborhood. *Variable neighborhood descent* (VND) extends the classic local search by systematically changing the neighborhoods and has been studied in the context of the *variable neighborhood search* (VNS) [46, 47]. In contrast to [46, 47], we describe here a slightly more general variant of the VND that is not restricted to the best improvement strategy but also allows the application of the next improvement rule.

Algorithm 2.4: Variable Neighborhood Descent

Input: solution s of a COP instance, neighborhoods $\mathcal{N}_1, \ldots, \mathcal{N}_{l_{\max}}$

```
1 l ← 1;
2 while l ≤ l_max do
3     if ∃s' ∈ N_l(s) with f(s') > f(s) then
4         s ← choose s' ∈ N_l(s) with f(s') > f(s);
5         l ← 1;
6     else
7         l ← l + 1;
```

Algorithm 2.4 shows the VND for a COP where the objective function has to be maximized. The input of the procedure consists of an initial solution s and a set $\{\mathcal{N}_1, \ldots, \mathcal{N}_{l_{\max}}\}$ of neighborhoods. The VND starts with the first neighborhood $\mathcal{N}_1$ and performs local search as in Algorithm 2.3. After a locally optimal solution has been reached w.r.t. neighborhood $\mathcal{N}_1$, the neighborhood index l is increased by one and the second neighborhood $\mathcal{N}_2$ is considered next. If possible, we perform for neighborhood $\mathcal{N}_2$, or in general for $\mathcal{N}_l$, a single improving move and reset index l back to 1. If on the other hand the current incumbent solution is already locally optimal w.r.t. $\mathcal{N}_l$, we increase l by one. The algorithm terminates after no improving solution could be found in neighborhood $\mathcal{N}_{l_{\max}}$, which implies that the incumbent solution is locally optimal w.r.t. all given neighborhoods.

2.3.3 Greedy Randomized Adaptive Search Procedure

The *greedy randomized adaptive search procedure* (GRASP) [34, 35, 102] is a simple multistart metaheuristic shown in Algorithm 2.5. In each iteration a new solution is created by a randomized greedy heuristic which is then improved by a local search algorithm. The GRASP stops after reaching a termination criterion and returns the best found solution over all completed iterations.

It is essential that the solutions constructed in each iteration is with a high probability distinct from each other in order to provide diverse starting points for the typically deterministic local search. To achieve this the employed constructive heuristic selects

Algorithm 2.5: Greedy Randomized Adaptive Search Procedure

```
   Input: a COP instance
1  while termination criterion not met do
2      s ← construct a greedy randomized solution;
3      s ← apply local search on s;
```

the next element to extend the currently constructed solution in a greedy randomized way. For this purpose all feasible extensions of the current partial solution are ranked according to a greedy evaluation function. The best ranked extensions are collected in a so-called restricted candidate list. The element that is used to extend the current partial solution is then randomly selected from this list.

The size of the restricted candidate list controls the randomness of the construction and is next to termination criterion the main parameter of the GRASP. When the size is limited to one, then the construction corresponds to a pure greedy heuristic. In the other extreme, if the size of the restricted candidate list is not limiting, then a completely random solution is generated. To reach the COP's most valuable solutions frequently diverse but promising solutions have to be constructed for the subsequent local search. There are two common ways to limit the size of the restricted candidate list. Either the list's size is restricted by a quality threshold or by a fixed maximum number of elements.

2.3.4 Iterated Greedy

Iterated greedy (IG) metaheuristics [60, 107] are based on the observation that greedy constructive heuristics frequently select better extensions when already larger parts of the partial solution are fixed. Therefore, IGs repeatedly destroy random parts of the incumbent solution which are then repaired using a greedy constructive heuristic. The intention is that in the course of reapplying the destruction and reconstruction operations increasingly larger parts of an optimal or nearly optimal solution survive in the incumbent solution.

Algorithm 2.6: Iterated Greedy

```
   Input: solution s of a COP instance
1  while termination criterion not met do
2      s^P ← destruct parts of s;
3      s' ← construct complete solution from s^P;
4      s' ← apply local search on s';                    // optional
5      if s' is accepted then
6          s ← s';
```

Algorithm 2.6 outlines the procedure. The IG starts with an initial solution which is often obtained by the same greedy heuristic which is also employed in the construction

phase. This initial solution serves as a first incumbent solution. Each iteration begins with a destruction phase annulling parts of the incumbent solution. The result is a partial solution. The destruction phase is typically randomized and is the main diversification mechanism of the metaheuristic. In the following construction phase the partial solution is completed by a greedy constructive heuristic. Frequently, a local search algorithm is applied after the construction phase to further boost the IG's intensification. Afterwards, an acceptance criterion is evaluated to determine whether the newly generated solution replaces the incumbent solution. A trivial acceptance criterion is to replace the incumbent solution only if the newly obtained solution is better. If early convergence is an issue, then frequently a simulated annealing-like acceptance criterion is employed as a second source of diversification [107, 91]. Such an acceptance criterion allows with a certain probability that also a worse solution replaces the incumbent solution. The IG repeats the above steps until a given termination criterion has been reached and returns the best found solution.

2.3.5 Variable Neighborhood Search

The *general variable neighborhood search* (GVNS) [46, 47] is a metaheuristic that uses neighborhoods for its intensification and diversification phase. The employed key mechanism is the systematic change of neighborhoods which has been already discussed for the VND (see Section 2.3.2). As the name suggests, the GVNS is a generalized variant of the basic VNS [86].

Algorithm 2.7: General Variable Neighborhood Search

Input: solution s of a COP instance,
neighborhoods $\mathcal{N}_1^{\text{vns}}, \ldots, \mathcal{N}_{k_{\max}}^{\text{vns}}$ and $\mathcal{N}_1^{\text{vnd}}, \ldots, \mathcal{N}_{l_{\max}}^{\text{vnd}}$

```
while termination criterion not met do
    k ← 1;
    while k ≤ k_max do
        s' ← random element from N_k^vns(s);
        s' ← apply VND on s' using N_1^vnd, ..., N_{l_max}^vnd;
        if f(s') > f(s) then
            s ← s';
            k ← 1;
        else
            k ← k + 1;
```

The GVNS is sketched in Algorithm 2.7. The method takes as input an initial solution s which is frequently provided either by a constructive heuristic or a more involved approach. Moreover, a set of shaking neighborhoods $\{\mathcal{N}_1^{\text{vns}}, \ldots, \mathcal{N}_{k_{\max}}^{\text{vns}}\}$ for the diversification and a set of local search neighborhoods $\{\mathcal{N}_1^{\text{vnd}}, \ldots, \mathcal{N}_{l_{\max}}^{\text{vnd}}\}$ for the intensification is required.

Each major iteration starts by picking a random solution s' from the first shaking neighborhood $\mathscr{N}_1^{\text{vns}}$ of the so far best solution s. Afterwards, we apply a VND using the neighborhoods $\mathscr{N}_1^{\text{vnd}}, \ldots, \mathscr{N}_{l_{\max}}^{\text{vnd}}$ on this randomly chosen solution and replace s' by the resulting solution. This step differs for the basic VNS where we would perform a local search that considers a single neighborhood instead of a VND. If this new solution s' is better than s, then we have found a new incumbent solution and replace s by s' and repeat the above steps with the first shaking neighborhood. Otherwise, we perform the above steps using the next shaking neighborhood. A major iteration ends when the randomly picked solution of the shaking neighborhood $\mathscr{N}_{k_{\max}}^{\text{vns}}$ yields not a new incumbent solution after the application of the VND. The whole approach terminates when a given termination criterion has been reached.

The efficiency of the GVNS is mainly influenced by the number and order of the neighborhoods. Typically, the shaking and local search neighborhoods are different from each other and ordered by their cardinality. Moreover, the cardinality of the shaking neighborhoods is usually larger than the cardinality of the local search neighborhoods. The intention behind this is that randomly sampled solutions of the shaking neighborhoods should steer the algorithm to unexplored parts of $\mathscr{S}$. In addition, picking a random solution in a neighborhood can be implemented very efficiently.

2.4 Decision Diagrams for Optimization

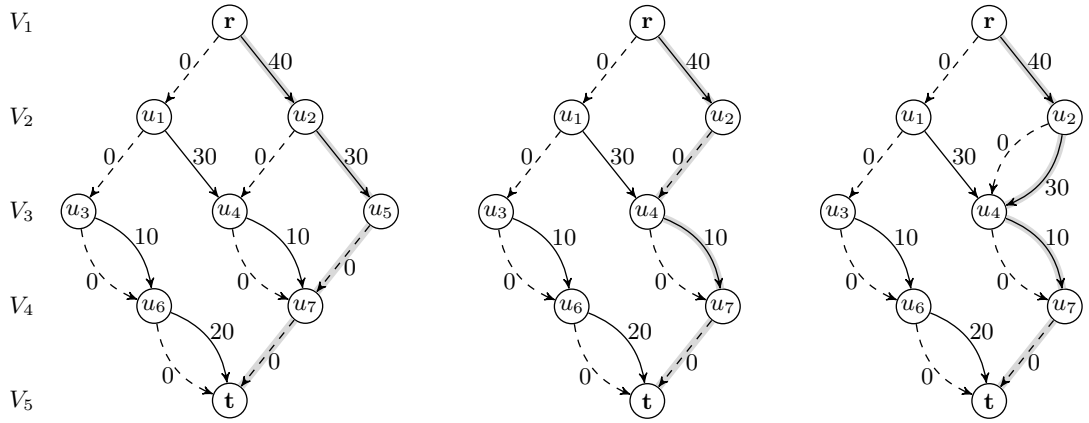
Decision diagrams (DDs) have been originally introduced in the field of electrical circuits and their formal verification [71, 58, 1, 20]. In about the last decade DDs have been recognized as a valuable tool in combinatorial optimization [3, 10, 23]. In particular, relaxed DDs provide new possibilities for modeling and solving parameterized discrete relaxations, enable new branching schemes, and allow advanced constraint propagation methods. For a comprehensive reading on DDs, their variants, applications and successes in optimization, we refer to the textbook by Bergman et al. [12].

DDs are strongly related to state graphs of a DP formulation. We formally define a DD as a rooted acyclic directed multi-graph $G = (V, A)$ with node set V and arc set A. The node set V is typically partitioned into layers $V = V_1 \cup \ldots \cup V_{n+1}$, where n is, in general, the number of decision variables of the underlying DP formulation. Each node $u \in V$ is associated with a state $\sigma(u)$ of the DP formulation. Arcs represent state transitions by assigning feasible values to decision variables of the problem. More specifically, an arc $a = (u, v) \in A$ is typically directed from a source node u in some layer V_i to a target node v in the subsequent layer V_{i+1}, $i \in \{1, \ldots, n\}$, and it is associated with the value $\text{val}(a)$ assigned to the i-th decision variable. Hence, the arc represents the transition from state $\sigma(u)$ to state $\sigma(v) = \tau(\sigma(u), \text{val}(a))$, where τ refers to the state transition function of the DP formulation. We assume that only arcs corresponding to feasible transitions are represented in the DD.

The first layer V_1 only contains a root node $\mathbf{r}$ corresponding to the initial state $\sigma(\mathbf{r})$ of the DP formulation with no decisions yet made. A single target node $\mathbf{t}$ at layer V_{n+1}

items	weight	value
1	3	40
2	4	30
3	2	10
4	5	20
capacity: 7		

Table 2.1: Description of a 0–1 knapsack instance.



(a) Exact DD (b) Restricted DD (c) Relaxed DD

Figure 2.1: Exact, relaxed, and restricted DDs for the 0–1 knapsack instance described in Table 2.1. Arcs directed from a node in layer V_i, $i \in \{1, 2, 3, 4\}$ specify whether item i is part of the solution. A solid arc indicates that the corresponding item is included in the solution, while a dashed arc represents that the respective item is not part of the solution. The numbers next to the arcs denote their length. Longest paths are highlighted in gray.

typically represents a final feasible state $\sigma(\mathbf{t})$ without any remaining decisions. Each path p from $\mathbf{r}$ to $\mathbf{t}$ consisting of n arcs represents a feasible solution.

Each arc $a \in A$ is associated with a length (or prize, cost etc.) $z(a) \geq 0$ that corresponds to the state transition's contribution to the objective function. The total path length is the objective value of the corresponding solution, and we are thus either looking for a longest or shortest path from $\mathbf{r}$ to $\mathbf{t}$ within the DD depending on whether we have maximization or minimization objective. We assume in the following a maximization objective.

A DD is shown in Figure 2.1(a) for the 0–1 knapsack instance described in Table 2.1. The underlying DP formulation is presented in Section 2.2.2 and consists of a binary variable for each item deciding whether an item is part of the solution or not. Each path in Figure 2.1(a) from node $\mathbf{r}$ to node $\mathbf{t}$ represents a feasible assignment of these variables.

The outgoing arcs of the root node **r**, for instance, assign the variable corresponding to the first item either to 0 or to 1 indicated by dashed and solid arcs, respectively. Consequently, if we follow the solid arc we include the first item in the solution and the total value up to this point is 40. If we select in addition the second item, we reach node u_5 and obtain a total value of 70. Since both of the items have a total weight seven which is equal to the capacity, no of the remaining items can be further added to the selected subset of jobs. Therefore, the path in the DD leading from u_5 to the target node **t** uses dashed arcs. This rightmost path is moreover the longest path of the DD and, hence, the subset consisting of the first two jobs is an optimal solution.

If the decision variables of the underlying DP formulation are as above all binary, the corresponding DD is frequently called a *binary decision diagram* (BDD). A consequence of the binary decision variables is that nodes of BDDs have at most two outgoing arcs. In contrast to BDDs, *multivalued decision diagram* (MDD) which we will use in Chapter 4 refer to DDs with more general finite domains.

An *exact DD* encodes by its paths exactly the set of all feasible solutions of the considered problem and it typically has an exponential size for hard problems. The literature describes two variants of more compact and, thus, practically more meaningful DDs that approximate exact DDs.

One possibility is to omit some arcs and nodes from an exact DD [11]. This implies that the resulting DD encodes only a subset of all feasible solutions and it is therefore called *restricted* DD. A longest path from **r** to **t** then corresponds to a feasible heuristic solution and its length is a lower (primal) bound on the optimal objective value. See for example Figure 2.1(b) that shows a restricted DD obtained from the exact DD of Figure 2.1(a) by removing node u_5 together with its incoming and outgoing arcs. Observe that the former longest path that selects the first two items is not anymore encoded. The longest path of the restricted DD is again the rightmost path representing the feasible but not optimal solution $\{1, 3\}$ with the total value of 50.

Restricted DDs are typically constructed in a top-down layer-by-layer fashion using some greedy criterion to select the states that are kept at each layer. Typically, the number of states at each layer is limited in order to obtain a restricted DD of bounded width. Note that this approach for obtaining a feasible heuristic solution closely corresponds to the well-known *beam search* metaheuristic [90].

Another possibility, is to construct a *relaxed* DD [3] that encodes a superset of all feasible solutions and represents a discrete relaxation of the original problem. A relaxed DD is in general obtained by superimposing states of the underlying DP state graph by merging nodes of an exact DD such that all original **r**–**t** paths are kept, but new ones can and typically do emerge. These new paths correspond to infeasible solutions. Consequently, the length of a longest **r**–**t** path in a relaxed DD is an upper (dual) bound on the underlying problem's optimal solution value. A relaxed DD for the 0–1 knapsack instance given in Table 2.1 is depicted in Figure 2.1(c). It is obtained from the exact DD by merging the nodes u_4 and u_5. In contrast to the exact DD, we are here able to select

also the third item after following twice a solid arc starting from **r**. The length of this path is 80 and, thus, higher as the optimal solution value of 70. It can be easily seen that this rightmost path does not correspond to a feasible solution as the total weight of nine exceeds our capacity of seven.

Two main approaches for compiling relaxed DDs of limited size have been proposed and these principles are to the best of our knowledge used in almost all so far published works where relaxed DDs are used to address COPs. The *top-down construction* (TDC) traverses the state graph layer-by-layer in a breadth-first manner starting from the root node **r**. The size of the DD is controlled by imposing an upper bound β, called width, on the number of nodes at each layer. For each state at a current layer, all successor states are obtained by considering each feasible state transition. For each such successor state, a corresponding node is created and inserted in the subsequent layer. If the size of the subsequent layer exceeds β, nodes of this layer are selected and merged until the size of the layer is reduced to β. The algorithm terminates when the last layer, and thus the target node **t**, is reached. Typically, some greedy criterion, e.g., based on the length of the longest paths from **r** to each node at the current layer, is used to select the nodes to be merged. The intention is to primarily merge nodes that are not likely part of a finally longest **r**–**t** path, in order to keep the obtained upper bound as low as possible. The merging itself must be done such that the new state associated with the merged node does not make any so far feasible extension of an original node towards a complete solution infeasible.

The second frequently used approach for constructing a relaxed DD is *incremental refinement* (IR). It starts with a trivial relaxed DD, e.g., a DD of width one, that has just one node in each layer. Then two major steps are repeatedly applied until some termination condition is fulfilled, e.g., a maximum number of nodes is reached. In the *filtering* step the relaxation represented by the DD is strengthened by removing infeasible arcs that cannot be part of any path corresponding to a feasible solution. Such arcs are determined by problem-specific calculations and deduction rules. Further, nodes that remain without any incoming arcs except **r** as well as non-terminal nodes without any outgoing arcs can be removed together with their remaining incident arcs. In the *refinement* step, nodes are split into pairs of new replacement nodes at the same layer in order to remove some of the paths that correspond to infeasible solutions. Again, the strategies for selecting meaningful nodes for splitting are problem-specific, but a frequent goal is to consider especially nodes at current longest paths that are involved in constraint violations.

Besides TDC and IR, Bergman and Cire [7] provide three algorithmic techniques, inspired by the concept of cutting planes from MILP, to improve the bounds of relaxed DDs. The first trims in each iteration a longest path until a considered longest path corresponds to a optimal solution. The second technique strengthens a relaxed DD by intersecting it with another relaxed DD. Lastly, the value extraction algorithm is presented that iteratively decreases the length of longest paths in the relaxed DD. In another work, Bergman and Cire [8] proposed to consider the compilation of a relaxed DD as an

optimization problem itself and investigated a MILP formulation. While this approach is useful for benchmarking different compilation methods on small problem instances, it is computationally too expensive for any practical application. Römer et al. [105] suggested a local search framework that serves as a more general scheme to obtain relaxed DDs. It is based on a set of local operations for manipulating and iteratively improving a DD, including the node splitting and merging from IR and TDC, respectively, and arc redirection as a new operator.

CHAPTER 3

Particle Therapy Patient Scheduling

This chapter addresses the midterm planning problem arising at the particle therapy treatment center MedAustron[1] located in Wiener Neustadt, Austria. The problem specification has been developed in several workshops together with the practitioners and stakeholders of MedAustron and colleagues from the University of Vienna. We then formalized the problem in terms of a mathematical model and created artificial benchmark instances that should represent the expected situation at the treatment center. Real world test data has not been available, as the facility was still under construction at the beginning of our project and by the end first patients have been treated, however, not yet yielding more detailed and representative information for scheduling.

In the course of our cooperation with MedAustron we published four scientific papers, an extended abstract, and a master's thesis [44] directly concerning this real word optimization problem. Moreover, another master's thesis [59] considered the problem of reacting on changes during the execution of a day's schedule. This chapter summarizes the mentioned four scientific papers which document the developments of our approach to tackle the midterm planning problem at MedAustron.

Our initial results have been presented at the *11th International Conference on the Practice and Theory of Automated Timetabling* (PATAT'16) and is published in the conference's proceedings:

> J. Maschler, M. Riedler, M. Stock, and G. R. Raidl. Particle therapy patient scheduling: First heuristic approaches. In *Proceedings of the 11th International Conference on the Practice and Theory of Automated Timetabling*, pages 223–244, Udine, Italy, 2016.

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

This paper presents the mathematical model, a constructive heuristic, and two first metaheuristics, namely a GRASP and an IG. In the later publications, we made some minor modifications in the problem definition regarding the structure of the treatments. The problem definition and all approaches presented in this chapter have been adapted to this new notation.

At the *12th Metaheuristics International Conference* (MIC'17) we presented and published in its proceedings a revised version of our IG metaheuristic:

> J. Maschler, T. Hackl, M. Riedler, and G. R. Raidl. An enhanced iterated greedy metaheuristic for the particle therapy patient scheduling problem. In *Proceedings of the 12th Metaheuristics International Conference*, pages 465–474, Barcelona, Spain, 2017.

For the *16th International Conference on Computer Aided Systems Theory* (EUROCAST'17) we first submitted an extended abstract and were invited after the presentation to submit a full paper:

> J. Maschler, M. Riedler, and G. R. Raidl. Particle therapy patient scheduling: Time estimation to schedule sets of treatments. In *Extended Abstracts of the 16th International Conference on Computer Aided Systems Theory (EUROCAST 2017)*, pages 106–107, Gran Canaria, Spain, 2017,
>
> J. Maschler, M. Riedler, and G. R. Raidl. Particle therapy patient scheduling: Time estimation for scheduling sets of treatments. In *Computer Aided Systems Theory – EUROCAST 2017*, volume 10671 of *Lecture Notes in Computer Science*, pages 364–372. Springer International Publishing, 2018.

We studied in this work a surrogate objective function that quickly predicts with reasonable precision the behavior of the sequencing part of our problem, allowing an improved distribution of the workload on the days.

Our final paper deals with an extended version of our problem and has been accepted for publication in the journal *International Transactions in Operational Research* (ITOR):

> J. Maschler and G. R. Raidl. Particle Therapy Patient Scheduling with Limited Starting Time Variations of Daily Treatments. *International Transactions in Operational Research*, 2018.

3.1 Introduction

Particle therapy is a relatively novel and highly promising option to provide cancer treatments. To this end, protons or carbon ions are produced by an ion source and are first accelerated by a *linear accelerator* (LINAC) and subsequently by either a cyclotron

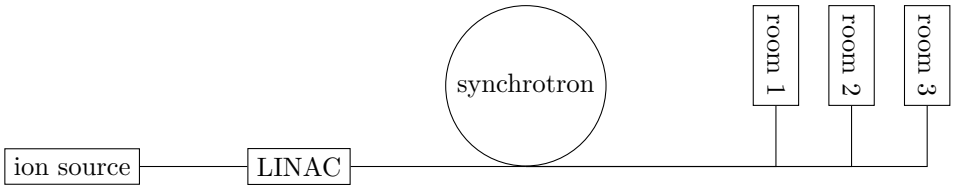
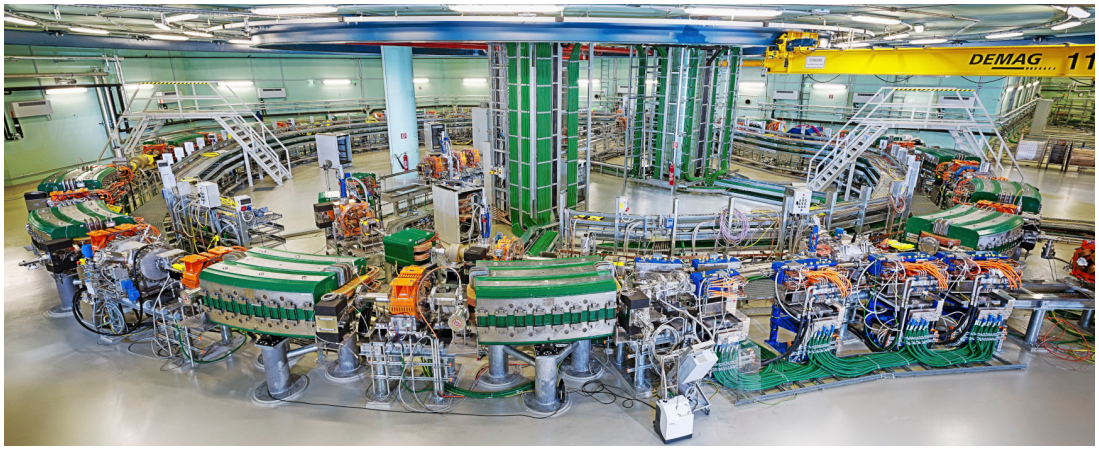


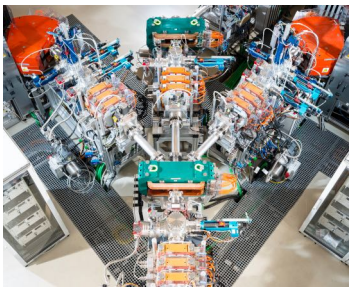
Figure 3.1: Layout of the particle therapy treatment center MedAustron.

or a synchrotron to around two-thirds of the speed of light. The resulting particle beam is then directed into one of up to five treatment rooms, where patients are irradiated. Since several tasks have to be completed in a treatment room before and after an actual irradiation, the usually single available beam is switched between the available treatment rooms in order to maximize the throughput of the facility. Consequently, the main challenge is to arrange the individual treatments in such a way that idle times on the particle beam are minimized. We consider here specifically the particle therapy treatment center MedAustron in Wiener Neustadt, Austria, which offers three treatment rooms. Figure 3.1 shows the layout of the facility. The different components of the layout are depicted in Figure 3.2. Each of the treatment rooms has individual capabilities regarding the flexibility of the particle beam. While the first room provides a fixed horizontal beam with protons and carbon ions, the second treatment room provides in addition irradiation with a fixed vertical beam. In the last of the three treatment rooms a gantry allows the most flexibility regarding the angle of the particle beam, but has the restriction that only proton beams are supported. A consequence of these different capabilities is that treatments are preassigned to the treatment rooms depending on the therapies' requirements.

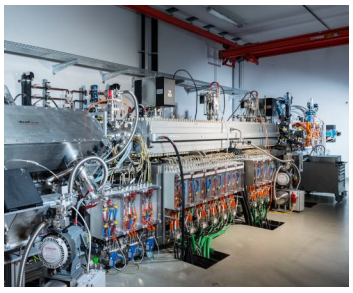
The *particle therapy patient scheduling problem* (PTPSP) addresses the midterm planning part of such a particle therapy treatment center. In PTPSP an effective plan has to be found for performing numerous therapies, each consisting of so-called *daily treatments* (DTs) provided on 8 to 35 succeeding days. Therapies have to start on Mondays or Tuesdays between an earliest and a latest allowed starting day. After a therapy is started, the number of DTs that are provided each week has to stay between a lower and an upper bound. Moreover, there is a minimal and a maximal number of days that are allowed to pass between two subsequent DTs, and there has to be a break from the treatment of at least two consecutive days each week. The DTs themselves model several consecutively performed tasks including the immobilization and the positioning of the patient, the actual irradiation, imaging, and the patient's exit from the treatment room. Therefore, DTs have resource requirements that vary with time according to the underlying tasks. A particularity is that each specific resource is required at most once for a consecutive time period. Moreover, each resource can only be used by one DT at a time. Every DT requires a patient resource, one of the three treatment rooms, and the particle beam. Certain DTs further involve special resources like a radio oncologist or an anesthetist.



(a) Synchrotron



(b) Ion source



(c) LINAC

(d) Treatment room

Figure 3.2: Photos of the components from MedAustron's layout shown in Figure 3.1. Source: press photos from www.medaustron.at. Copyright: Ettl / Thomas Kästenbauer.

Figure 3.3 depicts an example DT. In terms of the resource-constrained project scheduling literature (see for example [49]) DTs would be called activities with resource requests varying with time. The facility is usually open from Mondays to Fridays, but after holidays or maintenance tasks DTs are also performed on Saturdays. Whenever the treatment center is open, resources can have a regular availability period followed by an extended availability period in which they can be used, where the use of the latter induces (additional) costs. Furthermore, the availability of resources can be interrupted by so-called unavailability periods.

The aim of the PTPSP is to schedule a given set of therapies by determining days and times for all corresponding DTs while considering all operational constraints. The objective is to minimize the use of extended availability periods, while the therapies have to be completed as early as possible. Later in an extended problem variant, we consider that the DTs belonging to the same therapy should be planned roughly at the same time, in order to provide a consistent schedule for the patients. Ideally, the starting times of a therapy's DTs should not differ more than a half hour within each week. Between two

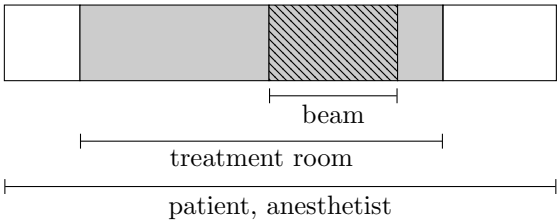


Figure 3.3: Exemplary DT that requires the patient and anesthetist resource for the whole processing time, while the resources representing the assigned treatment room and the particle beam are needed only for a specified part of the execution time.

weeks the starting times are allowed to differ by two hours. However, consistent starting times for DTs are not of direct medical relevance and, consequently, should not induce additional use of extended service periods or delay therapies.

We formalize PTPSP via an MILP model. However, even solving a strongly simplified version of the model turned out to be practically intractable. Therefore, we propose a therapy-wise construction heuristic, which acts in two phases by assigning first all DTs to days (day assignment) and then scheduling the DTs on each day (time assignment). Moreover, a GRASP and a first IG metaheuristic that are based on this construction heuristic are developed. Experiments indicate that this first IG yields superior results in comparison to the GRASP. This is mainly due to the fact that the IG preserves substantial parts of the solution from one iteration to the next, and consequently, poor decisions made especially in the first phase of the constructive heuristic can be corrected in the course of the iterations.

Nevertheless, this first IG does not exhaust its full potential: Moving DTs between days might require to reevaluate the start times of all DTs, and this is done by simply dropping all start times of the considered days. In addition, the IG's local improvement operator is based on applying a randomized version of the time assignment phase of constructive heuristic repeatedly for many times. Even though this local improvement operator is able to enhance solutions rather quickly this approach has the drawback that partly redundant work is repeatedly done, still yielding relatively similar solutions for the time assignments. Therefore, we revise this first IG and develop an improved metaheuristic. We propose novel destruction and construction methods that are able to keep relative timing characteristics of untouched DTs to a larger extent. Furthermore, we replace the so far rather simple local improvement operator by a local search method that considers a restricted DT exchange neighborhood. The new IG is compared to our previous IG and a variant of IG that uses the old destruction and construction phase combined with the new local search method. Our experiments clearly indicate that the IG with the new destruction and construction methods as well as the new local search yields substantially better results than the other two variants.

Afterwards, we focus on the decomposition of the PTPSP into a day assignment part

and a sequencing part. This decomposition makes the problem computationally more manageable as it allows us to separate the allocation of DTs to days with determining the DTs' starting times. However, both levels are dependent on a large degree. Especially, on the day assignment level we have to be aware of the behavior of the time assignment part. Hence, we provide a surrogate model that predicts the use of extended service windows given a set of DTs and a specific day. Experiments showed that the application within our IG metaheuristic allowed us to improve our previous results significantly.

Up to this point we considered PTPSP in its basic form, where the use of extended time and the therapies' finishing day has to be minimized. We shift our focus to the extended problem variant that covers the aspect that also the variation among the times at which therapies' DTs are provided has to be minimized. To this end, we reconsider our best performing IG approach and revise its components to be appropriate for the extended problem variant. The destruction and construction operator is further improved and also an enhanced construction heuristic for the initial solution is provided. The local search method is adapted to the extended problem formulation in that it alternately considers a DT exchange neighborhood and solves a LP model for determining updated nominal starting times. To investigate the performance of this final IG metaheuristic we adapt our first IG to the extended problem formulation. We assess the qualities of the individual components of the metaheuristic by gradually transforming our initial approach into the proposed one. Our revised metaheuristic shows considerable improvements on all considered benchmark instances.

The remainder of this work is structured as follows: After giving an overview of related literature in the next section, Section 3.3 provides a formal problem definition. Section 3.4 proposes a therapy-wise construction heuristic based on greedy principles and featuring a forward-looking mechanism to avoid too naive decisions. In Section 3.5 we further build upon this construction heuristic, proposing a GRASP and a first IG metaheuristic. This first IG is then revised in Section 3.6 and replacements for the employed local improvement, destruction, and construction operator are presented. Section 3.7 discusses a surrogate model that further improves our construction techniques. The extended problem formulation is considered in Section 3.8. To this end, we examine and make corrections to our previously developed IG metaheuristics. The conducted experiments are then discussed in Section 3.9. Finally, Section 3.10 concludes this chapter with an outlook on future work.

3.2 Related Work

Midterm planning for classical radiotherapy has attracted the focus of the scheduling community. A first attempt at automating this task has been made in 1993 by Larsson [70]. Then, for more than a decade no further contributions appeared. In 2006 interest in the topic grew again starting with the works of Kapamara et al. [63] and Petrovic et al. [97]. Afterwards, several heuristic as well as exact approaches followed. Heuristic techniques reach from GRASP [95, 96] and steepest hill climbing methods [62] to more advanced

techniques using GAs [93, 94]. Exact approaches consider different levels of granularity and rely on MILP models [24, 25, 26, 21]. Additionally, there are two PhD theses available dealing with the topic [85, 73].

The latest contributions focus on dynamic scenarios. Sauré et al. [108] consider a discounted infinite-horizon Markov decision process and solve it via linear programming with column generation to identify good policies for allocating available treatment capacity to incoming demand. Legrain et al. [72] introduce a hybrid method combining stochastic optimization and online optimization to determine better planning strategies. They also consider information on the future arrivals of patients to better estimate the expected resource utilization. Gocgun [41] additionally allows the cancellation of treatments and apply a simulation-based approximate dynamic programming algorithm to heuristically solve the problem. A literature review covering and categorizing works from 2000 to 2015 has been recently written by Vieira et al. [111].

All the references mentioned above consider a coarse scheduling scenario, i.e., they only assign treatments to days, but do not deal with sequencing within a day. This is due to the fact that the addressed practical applications feature radiotherapy with photons or electrons[2]. In these scenarios multiple LINACs are available, but for each of them only sequential processing is possible. Thus, the main issue is to assign treatments to appropriate machines. Therefore, there is no immediate need for fine-grained scheduling in these scenarios. The application we consider substantially differs in this respect since the availability of just a single accelerator whose beam can be directed to only one room at a time demands much more detailed planning to reach maximum throughput. Moreover, in the more widely applied photon and electron radiotherapy it is common to have long waiting lists associated with priorities determined by oncologists and algorithms need to select from that list when inserting new patients. In our scenario this is not required since the accepted patients must always be determined by physicians.

A scheduling problem concerning particle therapy is considered in Vogl et al. [112]. Although their problem is from the setting similar to ours, it differs in many details. While our emphasis is mainly on the throughput of the facility, i.e., on the scheduling of DTs under limited resource availabilities, the authors shift the subject more to the aspect of planning therapies including activities surrounding the core therapy. In particular, they have additional appointments that need to be provided either before or after a DT once a week. These appointments distinguish themselves from DTs as they can be supplied by different resources. In comparison to the PTPSP as described here, Vogl et al. assume that the resources are available on all days without any further restrictions. Moreover, their objective function differs substantially from ours: the aim is on minimizing the total idle time of the beam resource and the violations of time windows. Vogl et al. propose a multi-encoded GA and compare two solution decoding approaches.

[2]Men [85] considers proton therapy with a single particle beam and several rooms but also schedules only on the coarse level.

In a subsequent work Vogl et al. [113] extend their GA by a more complex crossover operator. In addition, Vogl et al. [113] present an *iterated local search* (ILS) metaheuristic which repeatedly first perturbates the so far best found solution followed by an application of local search. Perturbation is performed using two strategies alternatingly and as local search serves a VND using six neighborhoods. Their computational results showed that applying first the GA followed by the ILS starting from the best of the GA's solutions outperforms both individual metaheuristics.

The work by Riedler et al. [104] deals with strongly simplified variant of the time assignment part for a single day. Motivated by the fact that irradiation times are known exactly beforehand Riedler et al. [104] consider a *resource-constrained project scheduling problem* (RCPSP) for high time resolutions. Their solution approach is to relax the problem by partitioning time into so-called time-buckets, which are then iteratively refined until an optimal solution is found. We will present an overview over this approach in Chapter 5.

The problem considered by Horn et al. [55] is also inspired by the time assignment part for a single day. Each of their jobs requires one common resource during a part of its processing time and one of several secondary resources for the entire processing time. Such jobs model the essential aspect of our DTs, where the common resource corresponds to the beam and the secondary resources correlate with the rooms. They consider the minimization of the makespan as objective and provide an exact A* algorithm, a heuristic beam search, and a hybrid thereof. A price-collecting variant of this problem will be discussed in Chapter 4.

In principle the PTPSP can be viewed as highly specialized case of an RCPSP, which is a vast research area with many variants [49]. To consider all our requirements we need — in project scheduling terminology — disjunctive resources which have time-dependent capacities and overflow periods as well as multiple projects consisting of activities with resource requests varying with time, minimal and maximal time lags, release times, and deadlines; for an explanation of these aspects see [18]. Although each of these facets have been addressed in the literature, to our best knowledge no work exists considering all of them together. Of course, our objective also differs significantly due to the different domain. Research on the RCPSP can provide ideas for dealing with certain aspects in radiotherapy patient scheduling, but unfortunately none of the existing RCPSP variants is close enough to directly build upon it.

3.3 Problem Formalization

In the PTPSP a set of therapies $T = \{1, \dots, n_T\}$ needs to be scheduled on consecutive days $D = \{1, \dots, n_D\}$ considering a set of renewable resources $R = \{1, \dots, n_R\}$.

Each therapy $t \in T$ consists of a set of DTs $U_t = \{1, \dots, \tau_t\}$. For each therapy $t \in T$ we are given a minimal number n_t^{twmin} and a maximal number n_t^{twmax} of DTs that need to be performed per week, as well as a minimal number $\delta_t^{\min} \geq 1$ and a maximal number

$\delta_t^{\max} \geq \delta_t^{\min}$ of days that must separate two consecutive DTs. Between two weeks there have to be at least two days where no DT is performed. In addition, the maximum intended time difference of the starting times of the DTs within the same week and between two consecutive weeks is denoted with δ^{intraw} and δ^{interw}, respectively. The set of possible start days for each DT $u \in U_t$ is given by the subset $\{d_{t,u}^{\min}, \ldots, d_{t,u}^{\max}\} \subseteq D$ of days. For each DT $u \in U_t$ we are given a processing time $p_{t,u} \geq 0$ and a set of required resources $Q_{t,u} \subseteq R$. During the execution of a DT each resource $r \in Q_{t,u}$ is required for a part of the whole processing time specified by the time interval $P_{t,u,r} = [P_{t,u,r}^{\text{start}}, P_{t,u,r}^{\text{end}}) \subseteq [0, p_{t,u})$. We frequently write $P_{t,u,r}^{\text{dur}}$ as a shorthand for $P_{t,u,r}^{\text{end}} - P_{t,u,r}^{\text{start}}$, i.e., the duration resource r is required by DT u of therapy t.

The planning horizon is structured into a subset $D' \subseteq D$ of working days on which the treatment center is actually open and DTs can be scheduled on. The weeks covered by D are denoted by $V = \{1, \ldots, n_V\}$. Furthermore, let $\bigcup_{v \in V} D'_v$ be the partitioning of D' into n_V subsets corresponding to the weeks. For each working day $d \in D'$ we have a fundamental opening time $\widetilde{W}_d = [\widetilde{W}_d^{\text{start}}, \widetilde{W}_d^{\text{end}})$ that limits the availability of all resources on the considered day.

Each resource $r \in R$ is available on a subset $D_r^{\text{res}} \subseteq D'$ of the working days. On such days the availability of each resource is defined by a regular service time window $W_{r,d} = [W_{r,d}^{\text{start}}, W_{r,d}^{\text{end}}) \subseteq \widetilde{W}_d$ that is immediately followed by an extended service time window $\widehat{W}_{r,d} = [W_{r,d}^{\text{end}}, \widetilde{W}_d^{\text{end}}) \subseteq \widetilde{W}_d$. Moreover, for each resource $r \in R$ and each day $d \in D_r^{\text{res}}$, the availability of resource r may be interrupted by a set of unavailability intervals $\overline{W}_{r,d} = \bigcup_{w=1,\ldots,\omega_{r,d}} \overline{W}_{r,d,w}$ with $\overline{W}_{r,d,w} = [\overline{W}_{r,d,w}^{\text{start}}, \overline{W}_{r,d,w}^{\text{end}}] \subset W_{r,d} \cup \widehat{W}_{r,d}$. All these periods are assumed to be non-overlapping, and sorted according to increasing time.

We represent a solution for the PTPSP as a tuple (Z, S), where $Z = \{Z_{t,u} \in D' \mid t \in T,\ u \in U_t\}$ denotes the days at which the DTs are planned and $S = \{S_{t,u} \in \widetilde{W}_d \mid t \in T,\ u \in U_t,\ d = Z_{t,u}\}$ is the set of start times for all the DTs on the respective days. A solution is feasible if all resource availabilities, precedence relations, and the remaining operational constraints are respected.

A fundamental assumption is that it is in practice not difficult to find any feasible solution — we expect "enough" extended time of all the resources to be available. The background here is that once a patient is accepted for treatment it is ensured by all possible means that their treatment will take place according to all the defined requirements. What we aim for is to minimize the required extended time over all resources R while finishing each treatment as early as possible. These goals are considered in a single objective function by linearly combining the corresponding terms using the scalar weights γ^{ext} and γ^{finish}. More formally, we aim at minimizing

$$\gamma^{\text{ext}} \sum_{r \in R} \sum_{d \in D_r^{\text{res}}} \eta_{r,d} + \gamma^{\text{finish}} \sum_{t \in T} \left(Z_{t,\tau_t} - Z_{t,\tau_t}^{\text{earliest}} \right). \tag{3.1}$$

The first term of the objective function gives the total time used of the extended service time windows, where $\eta_{r,d} = \max(\{S_{t,u} + P^{\text{end}}_{t,u,r} - W^{\text{end}}_{r,d} \mid t \in T,\, u \in U_t, r \in Q_{t,u},\, Z_{t,u} = d\} \cup \{0\})$ for resource r and day d. The second term computes for each therapy t the deviation of the last treatment day from a lower bound $Z^{\text{earliest}}_{t,\tau_t}$, that represents a lower bound on the earliest possible day treatment t can be finished and is defined as

$$Z^{\text{earliest}}_{t,\tau_t} = \begin{cases} d^{\min}_{t,1} + \left(\left\lceil \frac{\tau_t}{n^{\text{twmax}}_t} \right\rceil - 1\right)(7 - n^{\text{twmax}}_t) + (\tau_t - 1) & \text{if } \delta^{\min}_t = 1 \\ d^{\min}_{t,1} + (\tau_t - 1)\delta^{\min}_t & \text{otherwise.} \end{cases} \tag{3.2}$$

Note that the definition of DTs stated here differs from the one given in our first paper [81], where DTs are composed of consecutively executed activities that are related with minimum and maximum time lags. The simplification here is motivated by the fact that in practice the possibility to have different minimum and maximum time lags between two activities is not expected to be exploited in midterm planning. Consequently, time lags may either be replaced by "dummy" activities of fixed length or, as we do here, the subdivision of DTs into activities can be replaced by the time intervals $P_{t,u,r}$ specifying at what times which resources are needed.

In our last work concerning the PTPSP [78], described in Section 3.8, we additionally aim at limiting the variations between the starting times of the DTs within each therapy. To this end, solutions are represented as a triple $(Z, S, \widetilde{S})$, where Z again denotes the assignments of DTs to days, S specifies once more the DTs' starting times on the respective days, and $\widetilde{S} = \{\widetilde{S}_{t,v} \in \mathbb{R}_{\geq 0} \mid \forall t \in T,\ v \in V\}$ corresponds to the set of nominal starting times of the therapies' DTs within the weeks. Next to minimizing the required extended time and the therapies' finishing days, the goal is to minimize the deviation of the DTs' starting times from the corresponding nominal starting times and to minimize the difference of the nominal starting times between weeks. The extended objective function is stated as

$$\gamma^{\text{ext}} \sum_{r \in R} \sum_{d \in D^{\text{res}}_r} \eta_{r,d} + \gamma^{\text{finish}} \sum_{t \in T} \left(Z_{t,\tau_t} - Z^{\text{earliest}}_{t,\tau_t} \right) + \gamma^{\text{intraw}} \sum_{t \in T} \sum_{u \in U_t \setminus \{1\}} \sigma^{\text{intraw}}_{t,u} + \gamma^{\text{interw}} \sum_{t \in T} \sum_{v \in V \setminus \{1\}} \sigma^{\text{interw}}_{t,v}, \tag{3.3}$$

where γ^{ext}, γ^{finish}, γ^{intraw} and γ^{interw} are scalar weights. The first two terms are defined as before. The third objective term gives the total excess of the allowed deviation of the DTs' starting times to their respective nominal starting times, where $\sigma^{\text{intraw}}_{t,u} = \max(|S_{t,u} - \widetilde{S}_{t,v}| - \delta^{\text{intraw}}, 0)$ with $v \in V : Z_{t,u} \in D'_v$. Each therapy's first DT is excluded from the above calculation since those are regarded in the specific situation at MedAustron as special. Finally, the last term computes the excess of the maximum intended time difference of the nominal starting times between two weeks and is calculated by $\sigma^{\text{interw}}_{t,v} = \max(|\widetilde{S}_{t,v} - \widetilde{S}_{t,v-1}| - \delta^{\text{interw}}, 0)$.

In the following we give a formal definition of the PTPSP by providing a MILP model. We start with the basic problem formulation that considers objective function (3.1). We

first focus on the upper level of assigning DTs to days. Afterwards, we give the additional constraints for assigning starting times to the individual DTs. Note, that these two levels are not independent. Nevertheless, the decomposition into a day assignment part and a scheduling part within days will be used within our approaches as it makes PTPSP more manageable in practice. Objective function (3.3) is considered next by a further extension of the model. To this end, we exchange the MILP's objective function and provide the additional variables and constraints for determining the nominal starting times and deviations from them.

3.3.1 Day Assignment

The MILP model for the day assignment uses the following variables. Binary variables $z_{t,u,d}$ that are one if DT u of therapy t is to be performed on day d, i.e., $Z_{t,u} = d$, and zero otherwise. Moreover, we have binary variables $y_{t,v}$ that are one if at least one of the DTs of therapy t is provided in week v and zero otherwise. In addition, variables $\eta_{r,d} \in \mathbb{R}_{\geq 0}$ give the amount of extended time used from resource r on day d. The latter variables clearly depend on the solution of the timing subproblem and their values are here estimated.

At the day assignment level, we relax the detailed timing structure of the DTs and consider only total resource availabilities on each day. Let $W^{\text{av}}_{r,d}$ denote the aggregated regular service time of resource r on day d and $\widehat{W}^{\text{av}}_{r,d}$ the available extended service time, i.e., the total time resource r is available on that day is $W^{\text{av}}_{r,d} + \widehat{W}^{\text{av}}_{r,d}$. Moreover, for better readability we denote by $D''_{t,u}$ for DT u associated with therapy t the set of working days between the DT's earliest and latest day, i.e., $D''_{t,u} = D' \cap \{d^{\min}_{t,u}, \ldots, d^{\max}_{t,u}\}$. The model reads as follows:

$$\min \gamma^{\text{ext}} \sum_{r \in \widehat{R}} \sum_{d \in D^{\text{res}}_r} \eta_{r,d} + \gamma^{\text{finish}} \sum_{t \in T} \left(\sum_{d \in D'} d z_{t,\tau_t,d} - Z^{\text{earliest}}_{t,\tau_t} \right) \tag{3.4}$$

$$\text{s.t.} \sum_{d \in D'} d\, z_{t,u+1,d} - \sum_{d \in D'} d\, z_{t,u,d} \geq \delta^{\min}_t \qquad \forall t \in T,\ \forall u \in U_t \setminus \{\tau_t\}, \tag{3.5}$$

$$\sum_{d \in D'} d\, z_{t,u+1,d} - \sum_{d \in D'} d\, z_{t,u,d} \leq \delta^{\max}_t \qquad \forall t \in T,\ \forall u \in U_t \setminus \{\tau_t\}, \tag{3.6}$$

$$\sum_{d \in D'_v} \sum_{u \in U_t} z_{t,u,d} \geq \min(n^{\text{twmin}}_t, |D'_v|) \cdot (y_{t,v} + y_{t,v+1} - 1) \qquad \forall t \in T,\ \forall v \in V \setminus \{n_V\}, \tag{3.7}$$

$$\sum_{d \in D'_v} \sum_{u \in U_t} z_{t,u,d} \leq n^{\text{twmax}}_t\, y_{t,v} \qquad \forall t \in T,\ \forall v \in V, \tag{3.8}$$

$$z_{t,u,d} + z_{t,u,d'} \leq 1 \qquad \forall t \in T,\ \forall v \in V \setminus \{n_V\},\ \forall d, d' \in D' : d \in \max\{D'_v\},\ d' \in \min\{D'_{v+1}\}, d' - d = 2, \tag{3.9}$$

$$\sum_{t \in T} \sum_{u \in U_t} P^{\text{dur}}_{t,u,r}\, z_{t,u,d} \leq W^{\text{av}}_{r,d} + \eta_{r,d} \qquad \forall r \in \widehat{R},\ d \in D^{\text{res}}_r, \tag{3.10}$$

$$\sum_{t \in T} \sum_{u \in U_t} P^{\text{dur}}_{t,u,r}\, z_{t,u,d} \leq W^{\text{av}}_{r,d} \qquad \forall r \in R \setminus \widehat{R},\ d \in D^{\text{res}}_r, \tag{3.11}$$

$$\sum_{d \in D'} z_{t,u,d} = 1 \qquad \forall t \in T,\ \forall u \in U_t, \tag{3.12}$$

$$z_{t,u,d} \leq y_{t,v} \qquad \forall t \in T,\ \forall u \in U_t,\ \forall v \in V,\ \forall d \in D'_v, \tag{3.13}$$

$$0 \leq \eta_{r,d} \leq \widehat{W}^{\text{av}}_{r,d} \qquad \forall r \in \widehat{R},\ \forall d \in D^{\text{res}}_r, \tag{3.14}$$

$$z_{t,u,d} \in \{0,1\} \qquad \forall t \in T,\ \forall u \in U_t,\ \forall d \in D''_{t,u}, \tag{3.15}$$

$$y_{t,v} \in \{0,1\} \qquad \forall t \in T,\ \forall v \in V. \tag{3.16}$$

Objective function (3.4) minimizes the use of extended time and prioritizes early finishing days. Inequalities (3.5) enforce that all the DTs of a therapy t are scheduled in the correct order and that the minimal number of required days $\delta^{\min}_t$ between two consecutive DTs is adhered. Similarly, inequalities (3.6) take care that the consecutive DTs of a therapy are scheduled no more than $\delta^{\max}_t$ days apart. The following two sets of constraints ensure for each therapy that the number of planned DTs stays within n^{twmin}_t and n^{twmax}_t per week. There are two special cases where the number of provided DTs might be lower than n^{twmin}_t. The first one is in the last week of therapies and the second one is if due to maintenance or public holidays the number working days within a considered week is less than n^{twmin}_t. Inequalities (3.9) require that if DTs can be scheduled on a Saturday and on the following Monday then treatments belonging to the same therapy may be scheduled on at most one of these days. This guarantees the required break of at least two days between weeks[3]. Subsequent inequalities (3.10) and (3.11) enforce that the amount of consumed resources does not exceed the amount of available resources on any day. Equations (3.12) assure that each DT is performed exactly once. Inequalities (3.13) link the x variables with the y variables. Inequalities (3.14) restrict the extended time that might be used. The domains of the z and y variables are specified by (3.15) and (3.16), respectively.

Again, be reminded that we relaxed timing aspects in the above model and so far only considered aggregated resource consumptions and availabilities. Consequently, this model on its own only provides a lower bound on the optimal solution value for the whole PTPSP.

[3]The facility is assumed to be always closed on Sundays.

3.3.2 Time Assignment

Let us now extend the above model by exactly modeling also the time assignments for the DTs. This allows us to calculate the correct values for the η variables. To ease the notation, let $S'_{t,u,d}$ be the set of all integral feasible starting times on day d allowing the corresponding DT $u \in U_t$, $t \in T$ to be performed without overlapping with one of the unavailability periods within its required resources' regular or extended service time. Moreover, we define $W'_{r,d}$ to be the set of integral time points at which resource r can be used on day d, i.e., $W'_{r,d} = (W_{r,d} \cup \widehat{W}_{r,d} \setminus \overline{W}_{r,d}) \cap \mathbb{Z}_{\geq 0}$.

The extension of the model is stated in terms of binary variables $x_{t,u,d,k} \in S'_{t,u,d}$ that are one if DT u of therapy t starts on day d at time point k, i.e., $Z_{t,u} = d$ and $S_{t,u,a} = k$, and zero otherwise. The extension of the model (3.4)–(3.16) reads as follows:

$$\sum_{k \in S'_{t,u,d}} x_{t,u,d,k} = z_{t,u,d} \qquad \forall t \in T,\ \forall u \in U_t,\ \forall d \in D''_{t,u}, \tag{3.17}$$

$$\sum_{\substack{t \in T,\, u \in U_t:\\ r \in Q_{t,u},\, d \in D''_{t,u}}} \sum_{\substack{k \in [b - P^{\text{end}}_{t,u,r},\, b - P^{\text{start}}_{t,u,r}]\\ \cap S'_{t,u,d}}} x_{t,u,d,k} \leq 1 \qquad \forall r \in R,\ \forall d \in D^{\text{res}}_r,\ \forall b \in W'_{r,d}, \tag{3.18}$$

$$\sum_{k \in S'_{t,u,d}} \left(k\, x_{t,u,d,k} + P^{\text{end}}_{t,u,r}\right) - W^{\text{end}}_{r,d} \leq \eta_{r,d} \qquad \begin{array}{r} \forall r \in \widehat{R},\ \forall d \in D^{\text{res}}_r,\ \forall t \in T,\\ \forall u \in U_t : r \in Q_{t,u},\ d \in D''_{t,u}, \end{array} \tag{3.19}$$

$$x_{t,u,d,k} \in \{0, 1\} \qquad \begin{array}{r} \forall t \in T,\ \forall u \in U_t,\ \forall d \in D''_{t,u},\\ \forall k \in S'_{t,u,d}. \end{array} \tag{3.20}$$

Equations (3.17) ensure that each DT gets assigned exactly one starting time at the DT's assigned day. For all other days all $x_{t,u,d,k}$ variables are forced to zero. Constraints (3.18) guarantee that each resource is used by at most one DT at a time. Inequalities (3.19) are used to calculate $\eta_{r,d}$, the required extended times of each resource $r \in R$, from the latest time it is in use by any DT. Remember that the sum over all $\eta_{r,d}$ variables appears in the objective function (3.4) and is to be minimized. The domains of the variables $x_{t,u,d,k}$ representing the starting times of the DTs are given in (3.20).

While we will see practical results for the MILP model of the day assignment relaxation (3.4)–(3.16), the time assignment extension was provided here only to specify the PTPSP in an exact way. Experiments very soon indicated that already solving the day assignment relaxation is computationally challenging. Solving the whole MILP model including the time assignments is clearly out of reach for the instances of practically relevant size. This would even hold already for a very crude time discretization. We will therefore focus on heuristic methods in this chapter.

3.3.3 Limiting Starting Time Variations

We provide here a further supplement to the model (3.4)–(3.20) in order to consider the extended problem formulation of PTPSP. We require the additional variables $\widetilde{S}_{t,v} \in \mathbb{R}_{\geq 0}$ that represent for therapy t the nominal starting time in week v. The excess of a starting time's allowed deviation to their respective nominal starting time from DT u belonging to therapies t is given by $\sigma^{\text{intraw}}_{t,u} \in \mathbb{R}_{\geq 0}$ variables. Moreover, variables $\sigma^{\text{interw}}_{t,v} \in \mathbb{R}_{\geq 0}$ describe the excess of the maximum intended time difference between the nominal starting times of two successive weeks $v-1$ and v of therapy t.

In order to minimize the fluctuation of the times at which the therapies' DTs are scheduled we replace in our MILP the objective function (3.4) by

$$\begin{aligned} \min\ & \gamma^{\text{ext}} \sum_{r \in \widehat{R}} \sum_{d \in D^{\text{res}}_r} \eta_{r,d} + \\ & \gamma^{\text{finish}} \sum_{t \in T} \left(\sum_{d \in D'} dz_{t,\tau_t,d} - Z^{\text{earliest}}_{t,\tau_t} \right) + \\ & \gamma^{\text{intraw}} \sum_{t \in T} \sum_{u \in U_t \setminus \{1\}} \sigma^{\text{intraw}}_{t,u} + \\ & \gamma^{\text{interw}} \sum_{t \in T} \sum_{v \in V \setminus \{1\}} \sigma^{\text{interw}}_{t,v}. \end{aligned} \tag{3.21}$$

In addition to the constraints (3.5)–(3.20) we consider the following constraints:

$$\left| \sum_{k \in S'_{t,u,d}} k\, x_{t,u,d,k} - \widetilde{S}_{t,v} \right| - \delta^{\text{intraw}} - (1 - z_{t,u,d}) \widetilde{W}^{\text{end}}_d \leq \sigma^{\text{intraw}}_{t,u} \qquad \begin{aligned} &\forall t \in T,\ \forall u \in U_t \setminus \{1\}, \\ &\forall d \in D''_{t,u},\ \forall v \in V : d \in D'_v, \end{aligned} \tag{3.22}$$

$$|\widetilde{S}_{t,v} - \widetilde{S}_{t,v-1}| - \delta^{\text{interw}} \leq \sigma^{\text{interw}}_{t,v} \qquad \forall t \in T,\ \forall v \in V \setminus \{1\}, \tag{3.23}$$

$$\widetilde{S}_{t,v} \geq 0 \qquad \forall t \in T,\ \forall v \in V, \tag{3.24}$$

$$\sigma^{\text{intraw}}_{t,u} \geq 0 \qquad \forall t \in T,\ \forall u \in U_t, \tag{3.25}$$

$$\sigma^{\text{interw}}_{t,v} \geq 0 \qquad \forall t \in T,\ \forall v \in V. \tag{3.26}$$

Inequalities (3.22) provide that the $\sigma^{\text{intraw}}_{t,u}$ variables are set to the excess of the allowed deviation from the respective DTs' starting time to their nominal starting time. Since nominal starting time are defined per therapy and week, we have to explicitly take care that the excess is computed with the nominal starting time of the week the DT is performed. To this end, we provide a constraint for each possible day and subtract a large constant (in this case $\widetilde{W}^{\text{end}}_d$) whenever a DT is not performed on the considered day making the left-hand side negative. Consequently, the constraint is only active for the

day the DT is planed. Inequalities (3.23) enforce that the $\sigma^{\text{interw}}_{t,v}$ variables represent the excess of the maximum intended time difference of the nominal starting times between two subsequent weeks. The domains of the $\widetilde{S}_{t,v}$, σ^{intraw}, and σ^{interw} variables are specified by (3.24), (3.25), and (3.26), respectively.

Taking into account that we minimize over the σ^{intraw} and the σ^{interw} variables all constraints involving absolute values can be linearized by replacing equations of the form $|x| - y \leq z$ by $x - y \leq z$ and $-x - y \leq z$. For instance, the constraints (3.23) are linearized to

$$\widetilde{S}_{t,v} - \widetilde{S}_{t,v-1} - \delta^{\text{interw}} \leq \sigma^{\text{interw}}_{t,v} \qquad \forall t \in T,\ \forall v \in V \setminus \{1\} \qquad \text{and}$$
$$\widetilde{S}_{t,v-1} - \widetilde{S}_{t,v} - \delta^{\text{interw}} \leq \sigma^{\text{interw}}_{t,v} \qquad \forall t \in T,\ \forall v \in V \setminus \{1\}.$$

The other constraints can be converted analogously.

3.4 A Therapy-Wise Constructive Heuristic

As a first, rather fast method to obtain heuristic solutions for the PTPSP, we propose the *therapy-wise construction heuristic* (TWCH) in the following. It acts in two phases, first assigning all DTs to days and afterwards scheduling the DTs on each day. The heuristic follows simple greedy principles but also uses a forward-looking mechanism to avoid getting trapped by making obviously poor decisions.

3.4.1 Day Assignment Phase

In the day assignment phase the heuristic iteratively selects one still unconsidered therapy and assigns days to its DTs sequentially, starting with the first DT. For each DT all feasible days between the earliest and latest starting day are evaluated w.r.t. the aggregated resource demands, the still available capacities, and the constraints imposed by the DT's predecessors. The DT is then assigned to the candidate day minimizing on the one hand the day at which the therapy is completed and on the other hand the expected additional use of extended service time windows for the current DT and all subsequent DTs.

Algorithm 3.1 shows this procedure in detail. It starts with a set of therapies T' to be scheduled, which initially corresponds to T, and an empty set G_d which is used to store the DTs assigned to each working day $d \in D'$. In each iteration of the while-loop a therapy t is selected and removed from T' according to a priority value determined by function therapy_priority($\cdot$). We will consider the calculation of this priority value later. The DTs $u \in U_t$ of the selected therapy t are then processed sequentially in the for-loop from Line 6 on.

For each DT a range of feasible days $\{d^{\text{earliest}}, \ldots, d^{\text{latest}}\}$ is determined first. This range is for all DTs at most $\{d^{\min}_{t,u}, \ldots, d^{\max}_{t,u}\}$. For all DTs for which the predecessor has been already assigned, i.e., all DTs except the first, the range of possible days can be restricted further. Line 10 and Line 16 exclude days from the range that are either to close or

Algorithm 3.1: TWCH's day assignment

Input: Set T' of therapies to consider

$G_d \leftarrow \emptyset \quad \forall d \in D'$;
while $T' \neq \emptyset$ **do**
  $t \leftarrow \arg\max_{t \in T'} \text{therapy\_priority}(t)$; *// ties are broken randomly*
  $T' \leftarrow T' \backslash \{t\}$;
  $d^{\text{last}} \leftarrow -1$; $v^{\text{last}} \leftarrow -1$; $n^{\text{tw}} \leftarrow 0$;
  for $u \leftarrow 1$ **to** τ_t **do**
    $d^{\text{best}} \leftarrow -1$; bestCost $\leftarrow \infty$;
    $d^{\text{earliest}} \leftarrow d^{\min}_{t,u}$; $d^{\text{latest}} \leftarrow d^{\max}_{t,u}$;
    if $u > 0$ **then**
      $d^{\text{earliest}} \leftarrow \max(d^{\text{earliest}}, d^{\text{last}} + \delta^{\min}_t)$;
      if $n^{\text{tw}} = n^{\text{twmax}}_t$ **then**
        $d^{\text{earliest}} \leftarrow \max(d^{\text{earliest}}, \max\{D'_{v^{\text{last}}}\} + 1)$;
      $v \leftarrow v \in V : d^{\text{earliest}} \in D'_v$;
      if $v^{\text{last}} \neq v$ **then**
        $d^{\text{earliest}} \leftarrow \max(d^{\text{earliest}}, d^{\text{last}} + 3)$;
      $d^{\text{latest}} \leftarrow \min(d^{\text{latest}}, d^{\text{last}} + \delta^{\max}_t)$;
      if $n^{\text{tw}} < \min(n^{\text{twmin}}_t, |D'_{v^{\text{last}}}|)$ **then**
        $d^{\text{latest}} \leftarrow \min(d^{\text{latest}}, \max\{D'_{v^{\text{last}}}\})$;
    foreach $d \in \{d^{\text{earliest}}, \ldots, d^{\text{latest}}\} \cap D'$ **do**
      if $\exists r \in R : P^{\text{dur}}_{t,u,r} + \sum_{(t',u') \in G_d} P^{\text{end}}_{t',u',r} > W^{\text{av}}_{r,d} + \widehat{W}^{\text{av}}_{r,d}$ **then** continue;
      if lookahead is feasible **then** obtain lookaheadCost **else** continue;
      extCost $\leftarrow \sum_{r \in Q_{t,u}} (\gamma^{\text{ext}} \cdot (\max(P^{\text{dur}}_{t,u,r} + \sum_{(t',u') \in G_d} P^{\text{dur}}_{t',u',r} - W^{\text{av}}_{r,d}, 0)) -$
       $\max(\sum_{(t',u') \in G_d} P^{\text{dur}}_{t',u',r} - W^{\text{av}}_{r,d}, 0)))$;
      finishCost $\leftarrow \gamma^{\text{finish}} \cdot (d - d^{\text{earliest}})$;
      if lookaheadCost + extCost + finishCost < bestCost **then**
        $d^{\text{best}} \leftarrow d$;
        bestCost $\leftarrow$ lookaheadCost + extCost + finishCost;
    if $d^{\text{best}} = -1$ **then** infeasible;
    $v \leftarrow v \in V : d \in D'_v$;
    if $v^{\text{last}} = v$ **then**
      $n^{\text{tw}} \leftarrow n^{\text{tw}} + 1$;
    else
      $v^{\text{last}} \leftarrow v$; $n^{\text{tw}} \leftarrow 1$;
    $d^{\text{last}} \leftarrow d^{\text{best}}$;
    $G_{d^{\text{best}}} \leftarrow G_{d^{\text{best}}} \cup \{(t, u)\}$;

too far from the predecessor. Lines 12 and 18 exclude days staying in conflict with the requirement that at least n_t^{twmin} and at most n_t^{twmax} DTs need to be provided per week. Moreover, Line 15 ensures that DTs are not performed on both Saturday and Monday. This guarantees in combination with the assumption that the treatment facility is generally closed on Sundays that between two weeks each therapy is paused for at least two consecutive days. Each working day in the obtained range is then further evaluated in the inner for-loop from Line 19 on. Days at which at least one resource required by the DT is not available in enough quantity are skipped by Line 20. Variables $W_{r,d}^{\text{av}}$ and $\widehat{W}_{r,d}^{\text{av}}$ denote here again the total available regular and extended time of resource r on day d.

For each remaining day the cost of assigning DT $u \in U_t$ to d are estimated by calculating the sum of the costs arising from using extended service time windows, a penalty cost for using later days and an estimation of the costs for assigning all successive DTs $u+1, \ldots, \tau_t$. The cost that originates from a single DT w.r.t. resource availabilities is the time the DT uses from extended service windows as calculated at Line 22. This simplistic estimation has been refined in Maschler et al. [84] and is described in Section 3.7. Line 23 computes how much selecting a specific day delays the whole therapy in relation to the second term of objective function (3.1). Line 34 finally assigns DTs u to the day with the lowest estimated cost, which is recorded in d^{best}.

To estimate the costs for assigning the successive DTs $u+1, \ldots, \tau_t$ we use a forward-looking mechanism that works almost analogously to the main algorithm's part from Line 6 to Line 33. The essential difference to Algorithm 3.1 is that in the lookahead we use for $u+1, \ldots, \tau_t$ the costs only from the first feasible day instead of considering the costs of all feasible days. This can be accomplished by omitting Line 21 and terminating the for-loop from Line 19 if line 26 is reached. The lookahead returns either the cost of assigning the DTs $u+1, \ldots, \tau_t$ to their first possible days, respectively, or the result that for at least one of the DTs no feasible day could be found. In the latter case we can remove the considered day from further consideration.

The performance of Algorithm 3.1 depends to a large extent on the therapy_priority$(\cdot)$ function. In preliminary tests we considered as this priority (a) the number of DTs τ_t, (b) the latest starting day for the first DT $-d_{t,1}^{\max}$ (with negative sign to give a DT with an earlier day higher priority), and (c) the first DT's total time required over all resources $\sum_{r \in Q_{t,1}} P_{t,u,r}^{\text{dur}}$, breaking ties randomly. With respect to our benchmark instances, see Section 3.9.1, our experiments clearly show that (b), the latest starting day for the first DT $-d_{t,1}^{\max}$, provides the best guidance for the heuristic. It appears reasonable that therapies having less flexibility w.r.t. their start should be given higher priority. Consequently, we will use this function in all our further investigations.

3.4.2 Time Assignment Phase

In the time assignment phase the scheduling of the DTs is done for each working day independently. For a considered day $d \in D'$, the heuristic selects a not yet scheduled DT from G_d and sets its start time as early as possible respecting the availabilities of all

required resources. Note that the time assignment presented here differs substantially from the original version proposed in [81] due to the simplifications in the structure of the DTs.

Algorithm 3.2: TWCH's time assignment

Input: day d and set G_d consisting of all assigned DTs

1 $C_r \leftarrow W^{\text{start}}_{r,d} \qquad \forall r \in R : d \in D^{\text{res}}_r$;
2 **while** $G_d \neq \emptyset$ **do**
3 $\quad (t,u) \leftarrow \arg\max_{(t,u)\in G_d} \text{DT\_priority}(t,u)$; *// ties are broken randomly*
4 $\quad G_d \leftarrow G_d \setminus \{(t,u)\}$;
5 $\quad S_{t,u} \leftarrow \max_{r\in Q_{t,u}}(C_r - P^{\text{start}}_{t,u,r})$;
6 $\quad$ **while** $\exists r \in Q_{t,u} \exists w \in \{1,\ldots,\omega_{r,d}\}\, [S_{t,u} + P^{\text{start}}_{t,u,r}, S_{t,u} + P^{\text{end}}_{t,u,r}) \cap \overline{W}_{r,d,w} \neq \emptyset$ **do**
7 $\quad\quad S_{t,u} \leftarrow \overline{W}^{\text{end}}_{r,d,w} - P^{\text{start}}_{t,u,r}$;
8 $\quad C_r \leftarrow S_{t,u} + P^{\text{end}}_{t,u,r} \qquad \forall r \in Q_{t,u}$;

Algorithm 3.2 shows the procedure in detail. The input is the considered day and the set G_d of assigned DTs as computed by Algorithm 3.1. For each resource r available on day d, a time marker C_r is initialized with the start of the regular service time window. This variable in general refers to the most recent point in time resource r has been used. All not yet scheduled DTs requiring resource r will be assigned a start time of at least C_r.

In each iteration of the main while-loop that starts at line 2 a DT $u \in U_t$ is selected and removed from G_d according to a priority function DT_priority$(\cdot)$. The tested realizations of this function are discussed later. We set first the DT's starting time $S_{t,u}$ to the earliest time s.t. every resource $r \in Q_{t,u}$ is not required earlier than C_r. At this point the starting time of the considered DT might still overlap with unavailability periods. Therefore, in the while-loop from line 6 we delay the starting time $S_{t,u}$ as long as there exists an overlapping unavailability period for a required resource. Afterwards, we have obtained either a feasible starting time or surpassed the end of the fundamental opening time. The latter case can be disregarded since we assume that there always exists sufficient extended time for scheduling any DT and, hence, satisfiability is not a practical issue. The last line of Algorithm 3.2 updates the corresponding time markers to include the last DT.

The solution quality of Algorithm 3.2 is influenced to a high degree by the greedy DT_priority$(\cdot)$ used for selecting the next DT to be considered. To this end we consider three different criteria where smaller values always indicating higher priorities:

(a) The idle time that emerges on the beam resource if DT $(t,u) \in G_d$ is considered next, i.e., the value $C''_{\text{B}} - C'_{\text{B}} - P^{\text{dur}}_{t,u,\text{B}}$ for the beam resource $\text{B} \in R$, where C'_{B} and C''_{B} are the values of the corresponding C_{B} variable before and after scheduling $u \in U_t$.

(b) The minimum time a resource required by the considered DT $(t, u) \in G_d$ leaves its regular service window, i.e., $\min_{r \in Q_{t,u}} W^{\text{end}}_{r,d}$.

(c) The ratio between the duration the beam required and the DT's total processing time, i.e., $P^{\text{dur}}_{t,u,\text{B}}/p_{t,u}$, where $\text{B} \in R$ denotes the beam resource.

Preliminary tests indicated that all the above criteria provide reasonable guidance with criterion (a) tending to yield on average better solutions than (b) and (c), but no single criterion dominates the others clearly. A problem, at least with respect to our benchmark instances, is that ties frequently happen, i.e., different DTs sometimes evaluate to the same priority criterion value. To counteract these ties, we define our actual DT_priority function via a lexicographic combination of all three above criteria: First, the value of criterion (a) is considered as priority. In case of a tie, criterion (b) is used, and if a tie happens again, the last criterion (c) is considered. Remaining ties are broken randomly. Note that the sign of all criteria values are inverted in order to obtain large priority values for DTs that should be preferred.

TWCH can be implemented in time $O(n_T \cdot \tau^2_{\max} \cdot d_{\max} \cdot Q_{\max} + |D'| \cdot n_T \cdot Q_{\max} \cdot \omega_{\max})$, where $\tau_{\max} = \max_{t \in T} \tau_t$ denotes the maximum number of DTs of any therapy, $d_{\max} = \max_{t \in T, u \in U_t}(d^{\max}_{t,u} - d^{\min}_{t,u})$ is the maximum number of days within which any DT must start, $Q_{\max} = \max_{t \in T, u \in U_t}(|Q_{t,u}|)$ represents the largest number of required resources by any DT, and $\omega_{\max} = \max_{r \in R, d \in D^{\text{res}}_r}(\omega_{r,d})$ specifies the maximum number of unavailability periods of all resources on a single day.

3.5 First Metaheuristic Approaches

As will be seen in the experimental results TWCH is relatively fast also on instances of practically relevant size. However, it obviously leaves room for improvements regarding solution quality, as some greedy decisions will in general not lead to an overall optimal solution. We therefore consider here two metaheuristic approaches that build upon TWCH: a GRASP and an IG metaheuristic.

3.5.1 GRASP

GRASP is a prominent metaheuristic building upon a construction heuristic and usually a local search component [102]. The basic idea is to apply a randomized variant of the construction heuristic independently many times, to locally improve each obtained solution, and to select an overall best solution as final one.

In our context, we randomize TWCH's day assignment by changing the way the next therapy to be scheduled is selected at line 3 of Algorithm 3.1 in a GRASP-typical fashion. Let p^{tmin} and p^{tmax} be the minimal and maximal priority value received from therapy_priority(t), $\forall t \in T'$, respectively. The next considered DT is chosen uniformly at random from the subset of T' with therapy_priority$(t) \geq p^{\text{tmax}} - \beta^{\text{gr-rand}} \cdot (p^{\text{tmax}} - p^{\text{tmin}})$. Parameter $\beta^{\text{gr-rand}} \in (0, 1)$ determines the allowed deviation from the highest priority

and thus controls the strength of the randomization. TWCH's original time assignment algorithm is then applied to all days to which DTs are assigned to, determining starting times S.

Afterwards, we apply as local improvement the following randomized multi-start heuristic individually to each day: TWCH's time assignment procedure is randomized by modifying line 3 to choose from the $k^{\text{rta-rand}}$ best DTs uniformly at random, with parameter $k^{\text{rta-rand}}$ controlling the strength of the randomization. This randomized construction is repeatedly applied until a schedule, not requiring extended times at the respective day, is found or no improvement has been achieved over the last $n^{\text{rta-noimp}}$ iterations. A starting time configuration inducing the smallest cost is finally kept.

3.5.2 Iterated Greedy

An IG [60] algorithm starts with an initial solution and then repeatedly applies a destruction phase annulling part of the solution, followed by a construction phase that completes the solution again, until a termination criterion is reached. The initial solution is usually obtained by applying a construction heuristic. The destruction phase removes randomly selected components from the incumbent solution, that are then reinserted by a greedy reconstruction method in the construction phase. Afterwards, an acceptance criterion is evaluated to determine whether the newly generated solution replaces the incumbent solution. Frequently, a local search algorithm is applied to the initial solution and after the construction phase to further boost the performance. The IG metaheuristic has many successful applications, in particular also in the domain of scheduling. See for example [107] where one of the first applications of IG to a permutation flowshop scheduling problem is described.

Our IG works as follows. TWCH as described in Section 3.4 is again used to create an initial solution. The destruction operator drops for $\beta^{\text{ig-dest}} \cdot n_T$ randomly selected therapies the assignment of all their DTs, i.e., invalidating their assigned days $Z_{t,u}$ and removing them from the sets G_d. Hereby, $\beta^{\text{ig-dest}} \in (0, 1)$ is an exogenous parameter, the destruction rate. The construction step is then performed in a straight-forward way by reapplying TWCH's day assignment for the set of removed therapies, warm-starting with the current sets G_d. Finally, TWCH's time assignment is applied from scratch to all working days for which G_d has changed in comparison to the original solution.

In addition, we also try to locally improve the obtained solution at each iteration by performing the day-wise multi-start randomized time assignment as described above for the GRASP. Our IG always accepts a new solution as new incumbent if it improves upon the previously best solution.

3.6 An Enhanced Iterated Greedy Metaheuristic

Experiments on our first two metaheuristic approaches, documented in Section 3.9.3, indicate that the IG yields superior results in comparison to the GRASP. This is mainly

due to the fact that the IG preserves substantial parts of the solution from one iteration to the next, and consequently, poor decisions made especially in the first phase of TWCH can be corrected in the course of the iterations. However, the IG from Section 3.5.2 does not exhaust its full potential: Moving DTs between days might require to reevaluate the start times of all DTs, and this is done by simply dropping all start times of the considered days. In addition, we used a local improvement operator within the IG that is based on applying a randomized version of the time assignment phase of TWCH repeatedly for many times. Even though this local improvement operator is able to enhance solutions rather quickly this approach has the drawback that partly redundant work is repeatedly done, still yielding relatively similar solutions for the time assignments.

We aim here at improving on the IG metaheuristic from Section 3.5.2. To this end, we propose a novel destruction and construction method that is able to keep relative timing characteristics of untouched DTs to a larger extent. Furthermore, we replace the so far rather simple local improvement operator by a local search method that considers a restricted DT exchange neighborhood. The initial solution is again provided by TWCH. This new IG outperforms the IG from Section 3.5.2 and a variant of the IG using the new local search method but still the originally proposed destruction and construction phase. In the following sections we discuss the new components of this enhanced IG metaheuristic.

3.6.1 Local Search

The design of the neighborhood used within the IG's local search component depends on several factors. As real world instances are expected to be quite large, the main challenge is to find neighborhoods that can be searched rather fast, still allowing to complete a reasonable number of iterations of the IG, while improving the solution significantly in most cases. To achieve this, we restrict ourselves to a local search method that is only able to modify the starting times of DTs, i.e., the day assignment is considered to be fixed. Hence, w.r.t. objective function (3.1) we are only able to improve on the objective function term that considers the use of extended service windows. A further consequence of this restriction is that the working days become independent, which allows us to define the neighborhood and perform the local search for each day separately.

In our scenario the DTs are heterogeneous regarding their time and resource requirements. Thus, moving DTs or exchanging the starting times of two DTs in a tightly scheduled day will lead in most cases to an infeasible solution. However, we can exploit the fact that each DT requires the beam resource exactly once to define a unique sequence of the DTs scheduled on a particular day. A solution is encoded by the sequence $(\pi_1, \ldots, \pi_n)$ resulting from sorting the DTs assigned to the currently considered day d (given by the set $\{(t,u) \mid t \in T, u \in U_t, Z_{t,u} = d\}$) in ascending order of the times from which on they use the beam resource B, i.e., according to $S_{t,u} + P^{\text{start}}_{t,u,\text{B}}$. On the encoded days we can then define classical neighborhoods for sequencing problems. To evaluate neighbors we have to decode the corresponding sequences of DTs to obtain actual starting times. Algorithm 3.3 shows this decoding for a given sequence $(\pi_1, \ldots, \pi_n)$ of DTs and a working

day $d \in D'$. The procedure starts by initializing each time marker C_r to the time the corresponding resource r becomes available. In the main loop each DT is assigned in the order of the given sequence to the earliest possible start time at which all resources are available. First, at Line 3 the DT's start time $S_{t,u}$ is set to the earliest possible time s.t. all required resources are used after their corresponding time marker. At this time, the considered DT might still overlap with unavailability periods. If this is the case, the DT is delayed in the inner while loop until all required resources become available. At Line 6 the time markers C_r are set to the times when the corresponding resources become free after the just scheduled DT. As for the TWCH's time assignment phase we omit here an explicit check that controls whether all DTs are performed before $\widetilde{W}_d^{\text{end}}$ since we assume that feasibility is not a practical issue.

Algorithm 3.3: Computation of starting times for a given sequence of DTs.

Input: A day d and a sequence of DTs $(\pi_1, \ldots, \pi_n)$

1 $C_r \leftarrow W_{r,d}^{\text{start}} \qquad \forall r \in R : d \in D_r^{\text{res}}$;
2 **for** $(t,u) \leftarrow \pi_1$ **to** π_n **do**
3 $\quad S_{t,u} \leftarrow \max_{r \in Q_{t,u}} (C_r - P_{t,u,r}^{\text{start}})$;
4 $\quad$ **while** $\exists r \in Q_{t,u} \exists w \in \{1, \ldots, \omega_{r,d}\} [S_{t,u} + P_{t,u,r}^{\text{start}}, S_{t,u} + P_{t,u,r}^{\text{end}}) \cap \overline{W}_{r,d,w} \neq \emptyset$ **do**
5 $\quad\quad S_{t,u} \leftarrow \overline{W}_{r,d,w}^{\text{end}} - P_{t,u,r}^{\text{start}}$;
6 $\quad C_r \leftarrow S_{t,u} + P_{t,u,r}^{\text{end}} \qquad \forall r \in Q_{t,u}$;

To obtain an effective neighborhood we have to take the problem structure into account. A fundamental property is that all DTs require the beam and one of the room resources. Moreover, the beam resource is used only during a part of the time the respective room resources are required. In a tight schedule the beam usually cycles between the three treatment rooms as in this way the emerging idle time on the beam resource is minimal and the throughput of the facility is maximized. If we interrupt this interleaved execution of DTs by removing a single DT from the sequence, then in general the resulting gap on beam resource cannot be fully closed by decoding the remaining DTs. Consequently, classic insertion moves in which a single DTs is removed and reinserted in another position of the encoded sequence will rarely improve already tight schedules. Exchanging the position of two DTs, however, circumvents this situation. Therefore, we consider a neighborhood structure based on such exchanges.

The DT exchange neighborhood is defined for a day d on a sequence $(\pi_1, \ldots, \pi_n)$ of DTs by considering all pairs of DTs π_i and π_j, where $1 \leq i < j \leq n$. A move in this neighborhood results in a new sequence $(\pi_1, \ldots, \pi_{i-1}, \pi_j, \pi_{i+1}, \ldots, \pi_{j-1}, \pi_i, \pi_{j+1}, \ldots, \pi_n)$ and is accepted if the decoded time assignment has a better objective value. The size of the neighborhood is $n(n-1)/2$.

To accelerate the local search procedure we restrict the DT exchange neighborhood to the most promising moves. That is, a move is only evaluated if both considered DTs are either adjacent in sequence π or it is likely that an exchange produces a tighter

Algorithm 3.4: Destruction and construction phases.

Input: A solution (Z, S)

$T' \leftarrow$ subset of T containing $\beta^{\text{ig-dest}} \cdot n_T$ randomly selected therapies;
remove day and time assignments for each therapy T';
apply TWCH's day assignment for T';
foreach $d \in D'$ **do**
  $G_d \leftarrow \{(t,u) \mid t \in T \setminus T', u \in U_t, Z_{t,u} = d\}$;
  $G'_d \leftarrow \{(t,u) \mid t \in T', u \in U_t, Z_{t,u} = d\}$;
  foreach $(t,u) \in G'_d$ **do**
    $(\pi_1, \ldots, \pi_n) \leftarrow$ sequence obtained by sorting G_d according to $S_{t,u} + P^{\text{start}}_{t,u,\text{B}}$;
    best_obj $\leftarrow \infty$; best_MS $\leftarrow \infty$; $\pi'_{\text{best}} \leftarrow ()$;
    for $i \leftarrow 1$ **to** $n+1$ **do**
      $\pi' \leftarrow (\pi_1, \ldots, \pi_{i-1}, (t,u), \pi_i, \ldots, \pi_n)$;
      schedule π' with Algorithm 3.3;
      obj $\leftarrow$ objective value of the current partial solution;
      MS $\leftarrow$ makespan of current day d;
      if obj $<$ best_obj $\vee$ (obj $=$ best_obj $\wedge$ MS $<$ best_MS) **then**
        best_obj $\leftarrow$ obj; best_MS $\leftarrow$ MS; $\pi'_{\text{best}} \leftarrow \pi'$;
    schedule π'_{best} with Algorithm 3.3;
    $G_d \leftarrow G_d \cup \{(t,u)\}$;

scheduled day. The latter criterion is based on the observation that scheduling two DTs requiring the same treatment room consecutively induces substantial idle time on the beam resource. Hence, we count how many adjacent DTs of $\pi(i)$ and $\pi(j)$ are requiring the same room as $\pi(i)$ and $\pi(j)$, respectively. A move is only considered further if this number does not increase with the exchange.

3.6.2 Destruction and Construction

The destruction and construction phase of the IG from Section 3.5.2 consists of removing the DTs of randomly selected therapies from the schedule, followed by applying TWCH's day assignment for the removed therapies and solving TWCH's time assignment from scratch. However, w.r.t. the local search algorithm from Section 3.6.1 discarding the whole time assignment during destruction and construction is disadvantageous since no parts of the old time assignment of an affected day are transferred to the new one. We overcome this drawback by replacing TWCH's day assignment with an insertion heuristic which preserves the sequence of unchanged DTs and inserts the removed ones in a greedy fashion. Note that this insertion heuristic is conceptually similar to the NEH algorithm of Nawaz et al. [87].

Algorithm 3.4 shows the used destruction and construction phase in detail. It starts by

invalidating the day and time assignment of $\beta^{\text{ig-dest}} \cdot n_T$ randomly selected therapies, where $\beta^{\text{ig-dest}} \in (0, 1]$ is the destruction rate. Afterwards, TWCH's day assignment is applied to reassign the DTs from the removed therapies to potentially new days. The insertion heuristic for the time assignment is defined in the foreach loop at Line 4 and is applied for each working day. The heuristic starts by initializing G_d to the set of DTs that have been assigned to day d and which have not been removed by the destruction phase. Analogously, G'_d is defined as the set containing all DTs assigned to day d that have been removed and for which a new starting time has to be found. Since all DTs in G_d have valid start times, we can define a unique sequence $(\pi_1, \ldots, \pi_n)$ by sorting the DTs according to the time they first require the beam resource. In each step a not yet considered random DT from G'_d is inserted at all possible positions of the sequence and scheduled using Algorithm 3.3. All of these $|G_d| + 1$ partial time assignments are compared and finally the best one is kept. To this end a partial solution is considered better if the objective value is smaller (i.e., the decoded sequence uses less extended time). In case of a tie we prefer the option with the smaller makespan. The rationale behind the latter criterion is that in particular after destruction many insertion points allow scheduling the sequence without use of extended service windows. Preferring a smaller makespan typically results in a tighter packed schedule and hopefully retains better options for the still to be inserted DTs.

3.7 Time Estimation for Scheduling Sets of Treatments

As already demonstrated in the problem formalization in Section 3.3, PTPSP naturally decomposes into the day assignment level in which DTs are assigned to days and the time assignment level that consists of finding starting times for the DTs. In other words, Z are the first level and S are the second level decision variables. Clearly, those two levels are dependent on a large degree. Nevertheless, this problem decomposition is beneficial because we can separate the detailed resource model from the remaining operational constraints. Moreover, if objective function (3.1) is considered in the time assignment level each day becomes independent and can be solved separately.

A central aspect of the day assignment is to find a well-paired allocation of DTs to days that allows a high throughput of the facility but causes as little use of extended availability periods as possible. Since determining $\eta_{r,d}$, the consumed extended time of resource r on day d, requires the precise knowledge of the starting times, the usage of the resource's service time windows for a given candidate set of DTs has to be estimated. For this reason, we use in the day assignment phase a modified version of our objective function (3.1) that replaces $\eta_{r,d}$ with the surrogate $\widehat{\eta}_{r,d} = \max(0, \widehat{\lambda}_{r,d} - W^{\text{av}}_{r,d})$, where $\widehat{\lambda}_{r,d}$ estimates the required time and $W^{\text{av}}_{r,d}$ again denotes the aggregated regular availability of resource r on day d. So far we used for $\widehat{\lambda}_{r,d}$ the trivial lower bound given by aggregated resource demands $\sum_{(t,u):t\in T,\, u\in U_t, r\in Q_{t,u},\, Z_{t,u}=d}(P^{\text{end}}_{t,u,r} - P^{\text{start}}_{t,u,r})$. Consequently, the day assignment frequently underestimated the resource consumption, which resulted in avoidable use of extended availability periods in the time assignment.

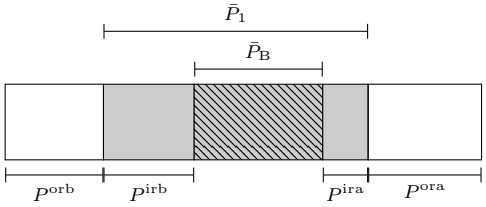


Figure 3.4: Durations P^{orb}, P^{irb}, P^{ira}, P^{ora}, and $\bar{P}_r$ depicted on an explanatory DT performed in treatment room 1. The gray area of the DT represents the part of the processing time that requires the room resource. The time for which the beam resource B is needed is illustrated by a hatched pattern. The remaining time of the DT when neither the beam nor room resource is required is shown in white. Note that P^{orb}, P^{irb}, P^{ira}, P^{ora}, and $\bar{P}_r$ are actually minimum and average durations over all considered DTs.

Here we focus on further improving the day assignment level by using a fast to compute, more accurate surrogate model for estimating $\widehat{\lambda}_{r,d}$ for the main bottleneck resources, the beam and the rooms. To this end, we study surrogate functions that consider more aspects of the problem at hand. The main goal is to predict the overall objective function contribution of the sequencing part with reasonable precision while being computationally fast enough to be used in our existing approaches.

3.7.1 Estimating the Makespan under Complete Resource Availability

In the following we first concentrate on estimating the makespan required for a given non-empty set G of DTs under the assumption that all required resources are available without any further restrictions. We start by determining estimations of the makespan for three special cases. Afterwards, an estimation for the general case is derived that is based on the estimation for these special cases. Let $n_r = |\{(t,u) \in G \mid r \in Q_{t,u}\}|$ be the number of DTs requiring resource $r \in \{1,2,3,\mathrm{B}\}$, where 1, 2, 3 represent the treatment rooms and B the beam, respectively. Furthermore, let

$$\bar{P}_r = \begin{cases} \frac{\sum_{(t,u)\in G}(P^{end}_{t,u,r} - P^{start}_{t,u,r})}{n_r} & \text{if } n_r > 0 \\ 0 & \text{else,} \end{cases} \tag{3.27}$$

be the average time resource $r \in \{1,2,3,\mathrm{B}\}$ is required by DTs in G. Moreover, let P^{irb} and P^{ira} be the minimum durations a room is required before and after the beam resource, respectively, and let P^{orb} and P^{ora} be the minimum times required by any DT before and after the usage of the room resource, respectively. Figure 3.4 illustrates these durations on a schematic representation of a DT.

In the first special case we assume that all DTs in G require the same room. Figure 3.5(a) shows an example with three DTs. Hence, w.r.t. the room resource all DTs have to be scheduled in a strictly sequential way. In addition, the beam will have substantial breaks.

In this case the makespan can be estimated using the total time the respective room resource is required and some constant offset for the tasks outside of the room, i.e., by

$$\max\{\bar{P}_1 n_1, \bar{P}_2 n_2, \bar{P}_3 n_3\} + P^{\text{orb}} + P^{\text{ora}}. \tag{3.28}$$

Observe that because only one room is used exactly one term of the maximum function in (3.28) is greater than zero.

The second special case supposes that the DTs are provided in two rooms. An example is provided Figure 3.5(b). DTs will be scheduled alternatingly between the two rooms. It can be assumed that the tasks apart the irradiation take in general longer than the irradiation itself. Consequently, there will be frequently breaks on the beam resource. In most cases, the makespan will be determined by the utilization of the room that is required the most. An estimation of the makespan for this special case is given by (3.28) again. In contrast to the previous scenario two terms of the maximum function are greater than zero.

The third special case assumes that the DTs are distributed evenly among the three treatment rooms. See Figure 3.5(c) for an example consisting again of three DTs. In such situations the rooms will be used in an interleaved way s.t. the beam cycles between all three rooms. In this way, the beam will typically be used most efficiently and it can be expected that the beam is used without idle time. The makespan can be estimated by the total time the beam resource is used plus a constant offset for the first and last scheduled DTs:

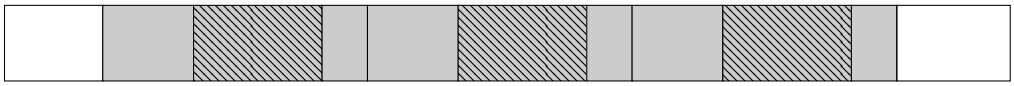
$$\bar{P}_{\text{B}} n_{\text{B}} + P^{\text{irb}} + P^{\text{ira}} + P^{\text{orb}} + P^{\text{ora}}. \tag{3.29}$$

In practice we will mostly have a mixture of the three discussed cases. A lower bound for the makespan can be derived by combining (3.28) and (3.29):

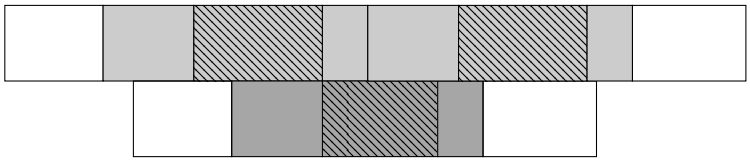
$$MS^{\text{LB}} = \max\{\bar{P}_1 n_1, \bar{P}_2 n_2, \bar{P}_3 n_3, \bar{P}_{\text{B}} n_{\text{B}} + P^{\text{irb}} + P^{\text{ira}}\} + P^{\text{orb}} + P^{\text{ora}}. \tag{3.30}$$

Equation (3.30) is a lower bound for the makespan since P^{orb}, P^{ora}, P^{irb}, and P^{ira} are the minimum durations that have to precede and follow the first and last use of the respective resources and the fact that the total resource requirement is a trivial lower bound. Basically, MS^{LB} assumes that there is a schedule without idle time on the resource that is used the most. Let $n_{\max} = \max_{r\in\{1,2,3\}} n_r$ and $n_{\min} = \min_{r\in\{1,2,3\}} n_r$. We can expect MS^{LB} to be a tight estimate if either $n_{\max} \geq n_{\text{B}} - n_{\max} - 1$, i.e., one room clearly dominates, or $n_{\max} \leq n_{\min} + 1$, i.e., the DTs are evenly distributed among the three rooms.

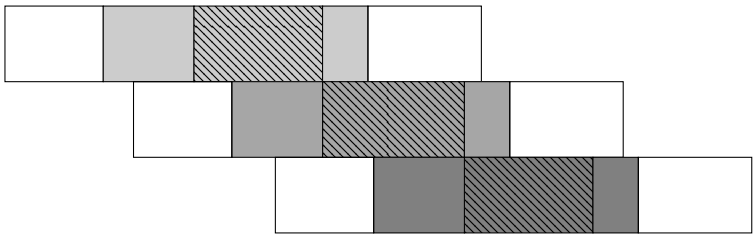
To strengthen the estimation also for cases in-between, we consider the simplified scenario in which all DTs have exactly the same timing and resource requirements, except that they are distributed among the three rooms. A good schedule would certainly cycle between all three rooms, but not to an extent that remaining DTs have to be scheduled sequentially in a single room.



(a) special case 1: all DTs require the same treatment room



(b) special case 2: the DTs are performed in two treatment rooms



(c) special case 3: the DTs are distributed evenly between all three rooms

Figure 3.5: Examples for the three special cases of the makespan estimation. In each example three DTs having the same timing characteristics are scheduled as densely as possible. DTs that require the same treatment room are depicted in the same row, while DTs performed in different treatment rooms are vertically shifted.

Let N_{123} be the maximal number of cycles between the three treatment rooms, such that all remaining DTs can be scheduled alternatingly between two rooms. In such a scenario, the following condition must hold:

$$n_{\max} - N_{123} - 1 = (n_{\min} - N_{123}) + (|G| - n_{\max} - n_{\min} - N_{123}). \tag{3.31}$$

The intuition of the formula above is to compare the number of DTs that remain in each room after cycling between all three rooms for N_{123} times. Note that the minus one represents the fact that the schedule might start and end with the room that is required the most. Equation (3.31) yields $N_{123} = |G| - 2n_{\max} + 1$. After excluding the corner cases where N_{123} becomes negative or larger than $n_{\min}$ we obtain

$$N_{123} = \min(n_{\min}, \max(0, |G| - 2n_{\max} + 1)). \tag{3.32}$$

We can now strengthen the estimation of the makespan by using for N_{123} cycles between the rooms the estimation for the third special case and for the remaining DTs the

estimation for the second special case as follows:

$$\begin{aligned} MS^{\mathrm{ES}} = \max\{&\bar{P}_{\mathrm{B}} n_{\mathrm{B}} + P^{\mathrm{irb}} + P^{\mathrm{ira}}, \\ &\bar{P}_1 n_1, \bar{P}_2 n_2, \bar{P}_3 n_3, \\ &3\bar{P}_{\mathrm{B}} N_{123} + \bar{P}_1 (n_1 - N_{123}), \\ &3\bar{P}_{\mathrm{B}} N_{123} + \bar{P}_2 (n_2 - N_{123}), \\ &3\bar{P}_{\mathrm{B}} N_{123} + \bar{P}_3 (n_3 - N_{123})\} + P^{\mathrm{orb}} + P^{\mathrm{ora}}. \end{aligned} \tag{3.33}$$

Notice that MS^{ES} is in contrast to MS^{LB} not a lower bound anymore.

3.7.2 Application of the Time Estimation

In this section the ideas developed in Section 3.7.1 will be used to obtain enhanced estimations for the total times the beam and each room is required. To this end, for a considered day $d \in D'$ let G be the set of all DTs assigned to day d. Since the beam and the rooms are normally available the whole day, we can assume that they have in general the same regular availability periods.

The total time the beam resource is required can be estimated almost analogously to (3.33) with the only difference that we have to disregard the time after the last DT has stopped using the beam. Thus, in the estimation, given by

$$\begin{aligned} \widehat{\lambda}_{\mathrm{B},d} = \max\{&\bar{P}_{\mathrm{B}} n_{\mathrm{B}} + P^{\mathrm{irb}}, \\ &\bar{P}_1 n_1 - P^{\mathrm{ira}}, \bar{P}_2 n_2 - P^{\mathrm{ira}}, \bar{P}_3 n_3 - P^{\mathrm{ira}}, \\ &3\bar{P}_{\mathrm{B}} N_{123} + \max(P^{\mathrm{irb}}, \bar{P}_1 \cdot (n_1 - N_{123}) - P^{\mathrm{ira}}), \\ &3\bar{P}_{\mathrm{B}} N_{123} + \max(P^{\mathrm{irb}}, \bar{P}_2 \cdot (n_2 - N_{123}) - P^{\mathrm{ira}}), \\ &3\bar{P}_{\mathrm{B}} N_{123} + \max(P^{\mathrm{irb}}, \bar{P}_3 \cdot (n_3 - N_{123}) - P^{\mathrm{ira}})\} + P^{\mathrm{orb}}, \end{aligned} \tag{3.34}$$

we have to subtract P^{ira} whenever the room resources are used for the estimation.

The total time the rooms are needed is estimated by

$$\begin{aligned} \widehat{\lambda}_{r\in\{1,2,3\},d} = \max\{&T_r n_r, \\ &3\bar{P}_{\mathrm{B}} N_{123} + \max(T_r (n_r - N_{123}), P^{\mathrm{irb}} + P^{\mathrm{ira}})\} + P^{\mathrm{orb}}. \end{aligned} \tag{3.35}$$

In contrast to the beam resource we can only use the considered room for the prediction. We can strengthen the estimation for the room resource that is used the most, i.e., for $r_{\max} = \arg\max_{r\in\{1,2,3\}} n_r$. This room is most likely the last one used and, hence, it is used at least as much as the beam resource. The strengthened estimation for room resource $r_{\max}$ is then given by

$$\widehat{\lambda}^*_{r_{\max},d} = \max\{\widehat{\lambda}_{r_{\max},d}, \bar{P}_{\mathrm{B}} n_{\mathrm{B}} + P^{\mathrm{irb}} + P^{\mathrm{ira}}\}. \tag{3.36}$$

3.8 An Iterated Greedy Metaheuristic for Limiting Starting Time Variations

Up to now all presented heuristic approaches considered the basic formulation of the PTPSP where solutions assign the therapies' DTs to days and to times at these days. Our approaches exploited the fact that once the allocation of DTs to days has been determined the time assignment decomposes to independent problems for each day. Here we shift our focus to the extension of PTPSP that covers in addition the aspect that the variation among the times at which therapies' DTs are provided has to be minimized. Besides starting days and starting times for every DT, solutions for this extension consist of a nominal starting time for each therapy and week. In contrast to before, the time assignment for the individual days cannot be regarded as independent but is coupled through the nominal starting times. Consequently, computing the nominal starting times simply as a third step, i.e., after all starting days and starting times have been fixed, will most likely result in poor solutions.

The aim of this section is to revise the IG metaheuristic from Section 3.6 in order to deal with this extended problem formulation effectively. To this end, we examine and make alterations to the most central components of the IG. For the initial solution as well as for the IG's construction phase we reuse TWCH's day assignment but replace the part for determining starting times with one that is more appropriate for the extended problem variant. Furthermore, the adapted local search method alternately considers the DT exchange neighborhood introduced in Section 3.6.1 and solves a LP model for determining updated nominal starting times. In the following sections we discuss the components in detail.

3.8.1 Initial Solution

TWCH is presented in Section 3.4 without the extension that the starting times should be close to their weekly nominal starting times and that the nominal starting times belonging to the same therapy should not differ too much. Basically, this construction heuristic acts in two phases, first assigning all DTs to days (day assignment) and afterwards determining the actual starting times of the DTs (time assignment). While the day assignment phase can be adopted unchanged, the time assignment phase has to be altered s.t. also the two new objective terms regarding the variation of the starting times are considered.

During the time assignment phase we have to find for all DTs starting times with as little use of extended service windows as possible that allow in addition nominal starting times that minimize the respective intra-week and inter-week objective terms. A changed property resulting from the extension of the PTPSP is that scheduling DTs in close succession might be suboptimal w.r.t. the two new objective terms. In fact, it might be necessary to have breaks between two consecutive DTs. In principle, our approach to consider the DTs in a certain order and schedule each DT as early as possible has to change to the one where we have to decide in addition the duration of an optional gap between each pair of successive DTs. However, in practice the facility should be used at

full capacity and, thus, it can be assumed that in general adding significant gaps between DTs immediately yields additional use of extended service time windows which results in a worse objective value. Therefore, we restrict the construction heuristic and our overall approach to solutions without gaps that are not induced by resource availabilities.

Our approach for the time assignment part consists of two components executed in an interleaved way, one for scheduling the DTs on a considered day and one for setting and adapting the nominal starting times. To this end, the working days are processed in chronological order, starting with scheduling the DTs assigned to the first working day. After all starting times for a day have been determined the nominal starting times of every considered therapy t in the current week v are updated as follows. Therapies' first DTs are ignored as they are excluded in the intra-week objective term. For each therapy's second DT we set $\widetilde{S}_{t,v}$ to $S_{t,2}$. For subsequent DTs u' assigned to the therapy's first week, the nominal starting time is set to the value that minimizes $\sum_{u=2}^{u'} \sigma_{t,u}^{\text{intraw}}$. Determining this value corresponds to finding the minimal value of a continuous piecewise linear function where the slope of the segments are multiples of γ^{intraw}. For DTs u' belonging to a therapy's second week and onward the nominal starting time $\widetilde{S}_{t,v}$ of the current week v is set to the value that minimizes $\sigma_{t,v}^{\text{interw}} + \sum_{u \in U_t | u \leq u', Z_{t,u} \in D'_v} \sigma_{t,u}^{\text{intraw}}$. To this end, the nominal starting time of the previous week is considered as fixed. Note that for this reason the determined nominal starting times might not be optimal.

TWCH's component for scheduling DTs within a day as presented in Section 3.4.2 repeatedly places a not yet scheduled DT, selected using a priority function, as early as possible in the schedule until all DTs have been planned. The priority of the DTs is determined by a lexicographic combination of three criteria that consider the idle time that emerges on the beam resource, the earliest end of a regular service window from a required resource, and the ratio between the time the beam is required and the total processing time of the respective DT. These greedy criteria provide in practice reasonable performance w.r.t. minimizing the use of extended service windows, while yielding short processing times. Extending this lexicographic combination of criteria to respect also the intra-week and inter-week objective terms is, however, not promising. The main difficulty is to balance between generating a tight schedule and prioritizing DTs that are close to their nominal starting time. While concentrating too much on the former aspect causes many deviations to the respective nominal starting times, focusing too much on the latter results often in extensive use of extended service windows.

To obtain a heuristic for scheduling DTs within a day that performs well on the extended problem formulation we shift the focus from which DT to schedule next to inserting DTs within the order of already scheduled DTs. A straightforward way to this would be to process the DTs assigned to a day in a particular order and test for each DT all positions of the already scheduled DTs. Finally, the DT is inserted in the position yielding the smallest objective value. This technique is analogous to the classic NEH algorithm by Nawaz et al. [87] for the *permutation flow shop problem* (PFSP). However, preliminary experiments have shown that the performance of this simple insertion heuristic is worse compared to our original approach. The main reason is that the insertion of DTs

postpones already inserted DTs which might end up at a quite different time they have been originally inserted. Hence, it makes sense to reevaluate the positions of already inserted DTs. In the PFSP literature several heuristics have been proposed that extend NEH with reinsertions. We adapt here the FRB3 heuristic from Rad et al. [99], that reevaluates and possibly reinserts after each insertion all already scheduled jobs.

Algorithm 3.5: FRB3 for the PTPSP

Input: day d

1 $\pi \leftarrow ()$;
2 let $(\beta_1, \ldots, \beta_n)$ be the DTs to schedule in decreasing order of $p_{t,u}$ with $\beta_i \in \{(t,u) \mid t \in T, u \in U_t, Z_{t,u} = d\}$ for all $i = 1, \ldots, n$;
3 **for** $i \leftarrow 1$ **to** n **do**
4 insert DT β_i in π resulting in the smallest objective value and in case of ties the smallest makespan;
5 **for** $j \leftarrow 1$ **to** i **do**
6 extract and reinsert DT π_j at the position that results in the smallest objective value and in case of ties in the smallest makespan;

Algorithm 3.5 shows FRB3 adapted for scheduling DTs on a considered day d. It starts with an empty sequence π and considers then the DTs assigned to day d in the order of the largest processing times. We use here the same ordering as Rad et al. [99] applied for FRB3 on the PFSP with the $C_{\max}$ (makespan) objective. Although our objective is quite different, preliminary experiments have shown that sorting according to the largest processing times is indeed effective. At Line 4, we test scheduling the current DT before each already scheduled DT and after the last one. The current DT is then inserted in the most promising position. In general, inserting a DT at position $l < i$ delays the starting times of the DTs $\pi_{l+1}, \ldots, \pi_i$, while the starting times of the DTs $\pi_1, \ldots, \pi_{l-1}$ remain unchanged. Afterwards, the position of all already scheduled DTs within π are reevaluated and possibly reinserted to accommodate the newly inserted DT in a better way.

DTs are inserted at the best position w.r.t. our objective function. However, as function (3.3) assumes a complete solution, we evaluate an objective function tailored for the time assignment part, which is

$$\gamma^{\text{ext}} \sum_{r \in R} \max\Big(\{S_{t,u} + P^{\text{end}}_{t,u,r} - W^{\text{end}}_{r,d} \mid i \in \{1, \ldots, n\},\ (t,u) = \pi_i,\ r \in Q_{t,u},\} \cup \{0\}\Big) + \sum_{\substack{i \in \{1,\ldots,n\},\\ (t,u)=\pi_i,\ u>2}} \gamma^{\text{intraw}} \cdot \sigma^{\text{intraw}}_{t,u}. \quad (3.37)$$

This function determines for the current sequence π the use of extended service windows and the excess of the allowed deviation from the nominal starting times. While the former term is analogous to (3.3), the latter term requires further considerations. As

mentioned earlier, the nominal starting times are updated after all starting times have been determined for a considered day. Consequently, for a therapy's second DT (i.e., the first for which the time difference to the nominal starting time is relevant) and every therapy's subsequent first DT of a week we have not yet determined the corresponding nominal starting time. In the case where a DT is a therapy's second we can set $\sigma_{t,u}^{\text{intraw}}$ to 0, as the respective nominal starting time can be set to the same value as the starting time of the current DT without inducing any cost. For a therapy's first DT within a week, we suppose that the nominal starting time will not differ more than δ^{interw} compared with the nominal starting time of the previous week. Hence, we regard in such cases starting times that differ more as $\delta^{\text{interw}} + \delta^{\text{intraw}}$ to their respective nominal starting time of the previous week as excess. For all other cases, $\sigma_{t,u}^{\text{intraw}}$ is computed as in the problem definition. To summarize, we calculate $\sigma_{t,u}^{\text{intraw}}$ by

$$\sigma_{t,u}^{\text{intraw}} = \begin{cases} 0 & \text{if } u = 2 \\ \max(|S_{t,u} - \widetilde{S}_{t,v-1}| - \delta^{\text{interw}} - \delta^{\text{intraw}}, 0) & \text{if } Z_{t,u} \in D'_v \wedge Z_{t,u-1} \in D'_{v-1} \\ \max(|S_{t,u} - \widetilde{S}_{t,v}| - \delta^{\text{intraw}}, 0) & \text{otherwise.} \end{cases} \tag{3.38}$$

In case more than one position evaluate to the same value by (3.37), we insert the DT that has the smaller makespan. The rationale behind the latter criterion is again that in particular at the beginning of the algorithm many insertion points allow scheduling the sequence without use of extended service windows. Preferring a smaller makespan typically results in a tighter packed schedule and hopefully retains better options for the still to be inserted DTs.

3.8.2 Local Search

As already argued in Section 3.6.1, the main challenge is to obtain a local search method that is fast enough in order to complete a sufficient number of iterations of the IG, while still improving the solution significantly in most cases. Due to typically small flexibility within the therapy process, assigning one DT to another day usually entails that also the therapy's preceding and succeeding DTs have to be reassigned to new days. Consequently, changing the day assignment of DTs affects the time assignment of several days, which results in a local search operator that is computationally too expensive to be applied within an iterated approach. Therefore, we restrict ourselves to a local search method that focuses on the time assignment part, i.e., the allocation of the therapies' DTs on days is assumed to be fixed. In the basic problem formulation without nominal starting times and the corresponding extensions this restriction allowed us to apply the local search for each day independently. The much smaller neighborhoods have shown to be crucial to receive adequate computation times for the local search component to be integrated within the IG. The presence of the therapies' nominal starting times and the extended objective function, however, links the starting times of the therapies' DTs: If we find a better starting time for a DT on a certain day, then we might also find a better

nominal starting time, which again induces improvements on days that have been at a local optimum. To obtain a fast local search component we consider first the nominal starting times as fixed and optimize the DTs' starting times until a local optimum w.r.t. a neighborhood for each individual day is reached. Afterwards, we update the nominal starting times and again repeat the first step until no further improvements are achieved. These two steps are alternated until no improvement can be achieved. In the following we explain both components in more detail.

We again encode solutions by the sequence $(\pi_1, \ldots, \pi_n)$ resulting from sorting the DTs assigned to the currently considered day d, given by the set $\{(t,u) \mid t \in T, u \in U_t, Z_{t,u} = d\}$, in ascending order of the times from which on they use the beam resource B, i.e., according to $S_{t,u} + P^{\text{start}}_{t,u,\text{B}}$. As in Section 3.6.1, we employ the DT exchange neighborhood that considers swapping the positions of a pair of DTs π_i and π_j, where $1 \leq i < j \leq n$. In other words, given a sequence $(\pi_1, \ldots, \pi_n)$ of DTs the neighborhood is defined of all sequences of the form $(\pi_1, \ldots, \pi_{i-1}, \pi_j, \pi_{i+1}, \ldots, \pi_{j-1}, \pi_i, \pi_{j+1}, \ldots, \pi_n)$. In order to evaluate a neighboring solution's objective value the corresponding sequence is decoded using Algorithm 3.3 to obtain actual starting times.

The examination of the neighborhood is computationally costly due to the fact that decoding sequences of DTs as well as evaluating the objective function are both expensive operations. Hence, we exploit several aspects in order to accelerate the search for improvements. The first speed-up is based on the observation that the starting times of the DTs $(\pi_1, \ldots, \pi_{i-1})$ are not affected from the move w.r.t. the original sequence. Consequently, Algorithm 3.3 is modified to consider for each move only the DTs following π_{i-1}. This requires, however, to store all possible states of the time markers C_r for each position of the original sequence. The next acceleration aborts the decoding of a neighbor within Algorithm 3.3 if the considered move yields no improvement with a high probability. In particular, this is the case if a move worsens the interleaving of the DTs or forces a DT to be placed after an unavailability period. The resulting delay produces as a consequence usually an additional use of extended service periods and in general an increased objective value. Therefore, we prematurely terminate the main loop at Line 10 after processing DT π_{j+k} if its newly determined starting time is larger than in the original sequence. To this end, offset k is chosen s.t. the corresponding DT is the first that requires the same room resources as π_i, thus, $k \in \{1, 2, 3\}$. We use this offset to assess whether DT π_i interleaves well with its successors. As in the FRB3 algorithm for the initial solution, it suffices also here to consider an objective function tailored to the considered day. Thus, we evaluate equation (3.37), where $\sigma^{\text{intraw}}_{t,u}$ can now be calculated as described in Section 3.3. Computing equation (3.37) from scratch can be done in $O(n_R + n \cdot n_Q + n)$ time, where n_Q is the maximal number of required resources by a DT. However, if we reuse the time markers C_r from Algorithm 3.3 the number of required steps decreases to $O(n_R + n)$.

After we reached a local optimum w.r.t. the considered neighborhood on each day, it might be that better suiting nominal starting times exist. Therefore, we assume now the set of all starting times S to be fixed and solve the following LP model to obtain

new optimal nominal starting times. In the model, we use $\widetilde{S}_{t,v}$ variables for the nominal starting times in $\widetilde{S}$ and nonnegative variables $\sigma^{\text{intraw}}_{t,u}$ and $\sigma^{\text{interw}}_{t,v}$ to state the intra-week and inter-week objective terms.

$$\min \gamma^{\text{intraw}} \sum_{t\in T} \sum_{u\in U_t\setminus\{1\}} \sigma^{\text{intraw}}_{t,u} + \gamma^{\text{interw}} \sum_{t\in T} \sum_{v\in V\setminus\{1\}} \sigma^{\text{interw}}_{t,v} \tag{3.39}$$

$$\text{s.t. } |S_{t,u} - \widetilde{S}_{t,v}| - \delta^{\text{intraw}} \leq \sigma^{\text{intraw}}_{t,u} \qquad \forall t \in T,\ \forall u \in U_t \setminus \{1\},\ \forall v \in V : Z_{t,u} \in D'_v, \tag{3.40}$$

$$|\widetilde{S}_{t,v} - \widetilde{S}_{t,v-1}| - \delta^{\text{interw}} \leq \sigma^{\text{interw}}_{t,v} \qquad \forall t \in T,\ \forall v \in V \setminus \{1\} : \exists u \in U_t(Z_{t,u} \in D'_v), \tag{3.41}$$

$$\sigma^{\text{intraw}}_{t,u} \geq 0 \qquad \forall t \in T,\ \forall u \in U_t, \tag{3.42}$$

$$\sigma^{\text{interw}}_{t,v} \geq 0 \qquad \forall t \in T,\ \forall v \in V, \tag{3.43}$$

$$\widetilde{S}_{t,v} \in \mathbb{R} \qquad \forall t \in T,\ \forall v \in V. \tag{3.44}$$

The model's objective function (3.39) corresponds to the objective function of the overall problem (3.3) restricted to the terms directly affected by the nominal starting times. Inequalities (3.40) enforce that the $\sigma^{\text{intraw}}_{t,u}$ variables are set to the excess of the maximum intended difference of the DTs' starting times to their respective nominal starting times. Finally, constraints (3.41) ensure that the $\sigma^{\text{interw}}_{t,v}$ variables attain the time difference that exceed the maximum intended time difference of the nominal starting times between consecutive weeks.

3.8.3 Destruction and Construction

In the destruction phase the DTs of randomly selected therapies are removed from the schedule. In the subsequent construction phase the DTs of the removed therapies are first assigned to days and afterwards suitable starting times are determined. To this end, we used in our first IG approach from Section 3.5.2 the proposed TWCH, which involved the time assignment part to be applied from scratch, completely ignoring the existing starting times of the kept DTs. In particular, w.r.t. the local search algorithm from Section 3.8.2 discarding the whole time assignment during destruction and construction is disadvantageous since much previous effort is wasted. Instead, we should try to transfer meaningful starting times as far as possible. To overcome this drawback we adopt in our improved IG, described in Section 3.6, the NEH insertion heuristic of Nawaz et al. [87] which preserves the sequence of unchanged DTs and inserts the removed ones greedily. Here we incorporate the extensions for NEH presented by Rad et al. [99].

In principle there are many options for the destruction phase. Since we keep the assignment of DTs to days fixed within the local search, it is especially important to allow such modifications during the destruction and construction phase. Due to the in general tight constraints on the day assignment level it can be expected that the

therapies' DTs are planned in close succession. Consequently, removing the assignments of single DTs within a therapy will have in general only limited effect as their assignments are usually determined by the remaining ones. In contrast, removing all assignments of some therapy allows much more flexibility like moving the therapy's start to another week. A further aspect is the greedy behavior of the TWCH's day assignment phase that acts like a first fit heuristic until it detects that another first fit assignment will induce use of extended service windows. At this point TWCH starts to actively delay and stretch the day assignments of the whole therapy as long it is beneficial w.r.t. the objective function. Thus, the destruction of the assignments of entire therapies allows TWCH revising poor decisions. Although one could think of different goal-driven selection strategies for therapies to remove, we choose to select random ones, since the destruction phase is our main source of diversification. To summarize, the deconstruction phase invalidates the day and time assignments of $n^{\text{ig-dest}}$ randomly selected therapies. To increase the robustness of the algorithm we do not keep $n^{\text{ig-dest}}$ fixed for all iterations, but sample a new value for $n^{\text{ig-dest}}$ from a discrete uniform distribution $\mathcal{U}^{\text{ig-dest}}$ at the beginning of every destruction phase.

The construction phase starts with an application of TWCH's day assignment on the destroyed therapies. Afterwards the respective DTs are inserted in a randomized order into the schedule. In the time assignment phase for the initial solution we use FRB3 for scheduling the DTs. After each insertion, all already scheduled DTs are reevaluated and possibly reinserted. In contrast to the initial solution the construction phase is executed for many times. Hence, the IG's construction phase is much more time critical and compared to FRB3 a less exhaustive approach is needed. Rad et al. [99] proposed among others the FRB4$_k$ algorithm which is conceptually between NEH and FRB3 in that it reconsiders only the k DTs around each inserted DT. FRB4$_k$ is based on the assumption that reevaluating the immediate neighbors has the largest effect. To be precise, we receive FRB4$_k$ if we modify the inner loop of Algorithm 3.5 s.t. it reevaluates the DTs at the positions $\max(1, l-k)$ to $\min(n, l+k)$, where l is the position of the previously inserted DT at Line 4. Moreover, unlike Algorithm 3.5 we start FRB4$_k$ within the construction phase not with an empty sequence π, but with the sequence resulting from sorting the not removed DTs according to the first time they use the beam resource (see Section 3.8.2). The nominal starting times are set as described in Section 3.8.1, followed by solving the model (3.39)–(3.44) after all starting times have been determined.

3.9 Computational Study

We start this section by describing the used benchmark instances and the used generation method. Then, the applied preprocessing techniques are discussed that aim at tighten the DTs' windows of possible starting days and to detect time periods during which resources cannot be used. Afterwards, we give details on the computational experiments of the proposed algorithms. In Section 3.9.3, we first assess the quality of TWCH's day assignment phase in comparison with an MILP approach. Moreover, we examine the final solutions of TWCH and of our first metaheuristic approaches. In the next section

we show the performance improvements of the enhancements introduced for the IG in Section 3.6. Later, in Section 3.9.5 the impact of employing the time estimation for scheduling sets of DTs within the IG's day assignment phase is analyzed. Finally, in Section 3.9.6 we consider the extended problem formulation and the corresponding IG metaheuristics that encompasses the improvements of all previous IG approaches.

3.9.1 Benchmark Instances

We created artificial benchmark instances related to real particle treatments and to the expected situation at MedAustron. These instances are available at http://www.ac.tuwien.ac.at/research/problem-instances. The main characteristic of an instance is its number of therapies n_T. We consider 5 instances for 50, 70, 100, 150, 200, and 300 therapies. In the used naming schema we encode first the number of therapies followed by a consecutive number. The modeled scenario considers a ramp-up phase of a few weeks at the beginning of a schedule after which the facility is used at full capacity followed by a wind-down phase near the end of the planning horizon. At full capacity the facility is able to perform around 60 DTs on each working day.

We use for all benchmark instances a time resolution of minutes, meaning that all specified times are multiples of a minute. The fundamental opening time $\widetilde{W}_d$ which encompasses all regular and extended service times is set for each working day to the interval $[0, 24 \cdot 60)$. Depending on the number of therapies considered by the instance, we determine in addition a regular opening time at the beginning of the fundamental opening time. Instances with less than 100 therapies have a regular opening time of the first 7 hours per day and for larger instances the regular opening time stretches over the first 14 hours of each working day. All resources' regular service windows are contained within this regular opening time. Consequently, smaller instances consider reduced regular availabilities which helps to keep them challenging.

We determine next the length of the time horizon n_D which is derived from the number of considered therapies. To this end, we assume that the first five days of each week are working days (i.e., Mondays to Fridays). The length of the time horizon n_D is based on an estimation of the total time required from the beam resource divided by a rough estimation on how much the beam resource can be used on each day and is computed by

$$n_D \coloneqq \left\lceil \frac{n_T \cdot E[\tau] \cdot E[p^{\mathrm{B}}]}{H^{\mathrm{open}} \cdot 0.7 \cdot 5} \right\rceil \cdot 7 + d^{\mathrm{buffer}}. \tag{3.45}$$

We denote with $E[\tau]$ the expected number of DTs per therapy and with $E[p^{\mathrm{B}}]$ the expected beam time per DT. The divisor consists of the regular opening time H^{open} which is in our case as mentioned above either $7 \cdot 60$ or $14 \cdot 60$. The next term in the divisor, 0.7, is a load factor to account the not yet considered unavailability periods and the fact that the beam resource cannot be used without breaks. The last value in the divisor and the subsequent multiplication 7 covers the fact that there are only 5 working days each week. Moreover, additional buffer days d^{buffer} are appended to ensure that

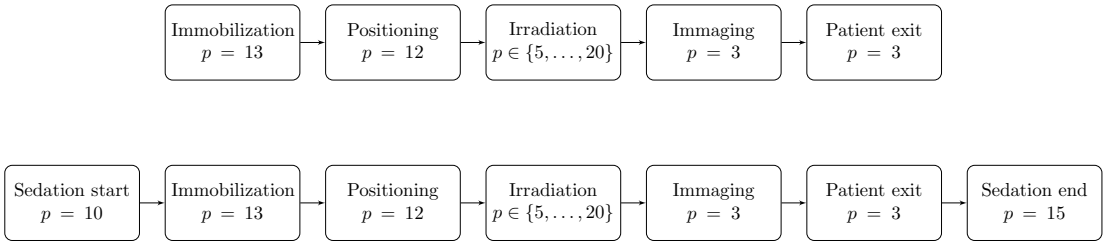


Figure 3.6: Activity structure modeled by DTs.

the length of the time horizon is not restrictive, i.e., delaying a therapy should never be prevented by the time horizon itself.

For each of the therapies the number of DTs is chosen uniformly at random from the interval $\{8, \ldots, 35\}$, reflecting the duration of real particle therapies. The period in which a therapy might start is assumed to have a fixed length of two weeks. We set for all therapies $n_t^{\text{twmin}} = 4$, $n_t^{\text{twmax}} = 5$, $\delta_t^{\min} = 1$ and $\delta_t^{\max} = 5$. Based on these values we calculate $d^{\text{buffer}} = 13$ that serves as an upper bound on the difference (in days) between the fastest and the slowest completion time of a therapy. Using this bound we determine the latest day by which the first DT of a therapy t might be provided: $d_{t,1}^{\text{latest}} := n_D - d^{\text{buffer}} - \lceil \tau_t / n_t^{\text{twmax}} \rceil \cdot 7 + 1$. The earliest day by which a therapy t can start, i.e., $d_{t,1}^{\min}$, is sampled uniformly from $\{1, \ldots, d_{t,1}^{\text{latest}} - 14\}$. Therapies are expected to have a window of two weeks during which they have to start. Therefore, we set $d_{t,1}^{\max}$ to $d_{t,1}^{\min} + 14$. Earliest and latest starting days of the subsequent DTs are not explicitly restricted, but are obviously implicitly limited by the remaining constraints.

As already mentioned in problem formalization, DTs model a sequence of consecutively executed activities. Most DTs capture five activities consisting of the immobilization, the positioning, the irradiation, the imaging, and the release of the patient. With a probability of 5% a therapy requires sedation and in that case its DTs consist of two additional activities. Figure 3.6 shows the two types of daily treatments together with the associated processing times in minutes. The processing times of the irradiation activities are drawn uniformly at random from $\{5, \ldots, 20\}$ as shown in the figure. For the processing times of the other activities we consider a spread of $\pm 20\%$, i.e., we choose from $\{\lfloor 0.8p \rfloor, \ldots, \lfloor 1.2p \rfloor\}$ uniformly at random. These choices are made per therapy and kept the same for all its DTs.

In the following we describe the resources associated with the activities. The main bottleneck in the considered application scenario is the beam resource which is required during the irradiation activity of each DTs. Its regular availability corresponds to the regular opening time on each working day. The availability of the three resources corresponding to the treatment rooms is set equivalently. A therapy's treatment room is chosen uniformly at random and is required for the positioning, irradiation, imaging, and patient exit activities. As mentioned above, some treatments require sedation which entails that an anesthetist is needed during all seven activities. The corresponding

resource has a regular service window spanning the first seven hours of each working day. Positioning activities of each therapy's first DT require the attendance of the oncologist responsible for the associated patient. Radio oncologists work in two shifts, the first shift spans the first two-thirds of the regular opening time and the second shift spans the last two-thirds of the regular opening time. In each shift five oncologists are available. For each therapy the associated oncologist is selected uniformly at random from both shifts. Moreover, the first treatment of each therapy needs to be provided either on Mondays or Tuesdays before noon. To model this we introduce an additional resource that spans the first seven of the regular opening time on these days. On each working day and resource there is unavailability period of 30 minutes in the middle of the regular opening time. For the instances featuring only seven operating hours these unavailability periods are placed at the end of the regular opening time. These 30 minutes serve for compensating delays that may arise.

When objective function (3.1) is considered, we use the weights $\gamma^{\text{ext}} = 1/60$ and $\gamma^{\text{finish}} = 1/100$. The intuition behind these values is as follows. Recall that the time resolution of the instances is in minutes. Weight γ^{ext} is set s.t. the use of one hour from an extended service time window corresponds to one unit in the objective function (3.1). The weight for the second objective term γ^{finish} reflects the fact that the completion of a therapy should usually be delayed if performing a DT entirely within extended time can be avoided. For the extended objective function (3.1) we apply the weights $\gamma^{\text{ext}} = 1/60$, $\gamma^{\text{finish}} = 1/100$, $\gamma^{\text{intraw}} = 1/600$, and $\gamma^{\text{interw}} = 1/600$. As already mentioned, providing the therapies' DTs within the allowed variance is not of medical relevance and is consequently a subordinate goal. Therefore, we set both γ^{intraw} and γ^{interw} to a tenth of γ^{ext}. A used hour of extended service windows is defined to be equally bad as ten hours of excess from the allowed variance between the starting times of the therapies.

3.9.2 Preprocessing

This section discusses three preprocessing techniques that are consecutively applied. The first aims at tightening the earliest and the latest starting day of DTs. This allows us to use the DTs' earliest and latest starting days to obtain a more accurate set of DTs that might get scheduled on a considered day. The two following methods regard the pruning of resource availabilities within working days. The basic idea is to identify time intervals during which resources are available but cannot be used by any DT. We then add or extend corresponding unavailability periods. Note that of the latter two methods no one dominates the other and, therefore, both are performed.

Preprocessing of Start Days

Each DT $u \in U_t$, $t \in T$ has a window of starting days specified by the earliest starting day $d^{\min}_{t,u}$ and the latest starting day $d^{\max}_{t,u}$. This preprocessing technique aims at tighten the bounds of these windows by taking into account the earliest and the latest starting day from previous DTs in combination with the minimal and the maximal number of allowed days between two consecutive DTs and the minimal and the maximal number

of DTs allowed per week. We explain the preprocessing of the earliest and the latest starting day of DTs by two separate algorithms.

Algorithm 3.6: Preprocessing of the earliest starting days

for $t \in T$ **do**
  $d_{t,1}^{\min} \leftarrow \min\{d \in D' \mid d \geq d_{t,0}^{\min}\}$;
  $v \leftarrow v \in V : d_{t,1}^{\min} \in D'_v$;
  if $|\{d \in D'_v \mid d \geq d_{t,1}^{\min}\}| \leq n_t^{\text{twmin}}$ **then**
    $d_{t,1}^{\min} \leftarrow \min\{D'_{v+1}\}$;
  for $u \leftarrow 2$ **to** τ_t **do**
    $d_{t,u}^{\min} \leftarrow \max(d_{t,u}^{\min}, d_{t,u-1}^{\min} + \delta_t^{\min})$;
    $d_{t,u}^{\min} \leftarrow \min\{d \in D' \mid d \geq d_{t,u}^{\min}\}$;
    $v \leftarrow v \in V : d_{t,u-1}^{\min} \in D'_v$;
    if $|\{d_{t,u'}^{\min} \in D_v \mid u' < u\}| \geq n_t^{\text{twmax}}$ **then**
      $d_{t,u}^{\min} \leftarrow \max(d_{t,u}^{\min}, \min\{D_{v+1}\})$;

Algorithm 3.6 considers each therapy $t \in T$ individually and starts by setting $d_{t,1}^{\min}$ to the next working day if $d_{t,1}^{\min} \notin D'$. Furthermore, $d_{t,1}^{\min}$ is advanced to the first working day in the next week if starting the therapy at day $d_{t,1}^{\min}$ would not allow the minimal number of DTs per week. For each subsequent DT u the earliest starting day is set to the maximum of the following three values: $d_{t,u}^{\min}$ as specified by the benchmark instance, $d_{t,u-1}^{\min} + \delta_t^{\min}$, and the first day in the next week if already n_t^{twmax} predecessors of the DT have the earliest starting day in the same week. Moreover, we ensure that the DT's earliest starting day is a working day by advancing $d_{t,u}^{\min}$ if it is not.

Algorithm 3.7: Preprocessing of the latest starting days

for $t \in T$ **do**
  $v \leftarrow v \in V : d_{t,1}^{\min} \in D'_v$;
  $d_{t,1}^{\max} \leftarrow \min(d_{t,1}^{\max}, \max\{D'_v\} - n_t^{\text{twmin}} \cdot \delta_t^{\min})$;
  $d_{t,1}^{\max} \leftarrow \max\{d \in D' \mid d \leq d_{t,1}^{\max}\}$;
  for $u \leftarrow 2$ **to** τ_t **do**
    $d_{t,u}^{\max} \leftarrow \min(d_{t,u}^{\max}, d_{t,u-1}^{\max} + \delta_t^{\max})$;
    $v \leftarrow v \in V : d_{t,u-1}^{\min} \in D'_v$;
    if $|\{d_{t,u'}^{\max} \in D'_v \mid u' < u\}| \geq n_t^{\text{twmin}}$ **then**
      $d_{t,u}^{\max} \leftarrow \min(d_{t,u}^{\max}, \max\{D'_{v+1}\} - n_t^{\text{twmin}} \cdot \delta_t^{\min})$;
    else
      $d_{t,u}^{\max} \leftarrow \min(d_{t,u}^{\max}, \max\{D'_v\} - (n_t^{\text{twmin}} - |\{d_{t,u'}^{\max} \in D'_v \mid u' < u\}|) \cdot \delta_t^{\min})$;
    $d_{t,u}^{\max} \leftarrow \max\{d \in D' \mid d \leq d_{t,u}^{\max}\}$;

Algorithm 3.7 attempts to tighten the latest starting days of DTs. Again each therapy $t \in T$ is processed independently. The method starts by setting the latest starting day of a therapy's first DT $d_{t,1}^{\max}$ to the closest working day smaller than or equal to the given $d_{t,1}^{\max}$ that allows at least n_t^{twmin} consecutive DTs in the current week. For each subsequent DT we set first the corresponding $d_{t,u}^{\max}$ to the minimum of $d_{t,u}^{\max}$ and $d_{t,u-1}^{\max} + \delta_t^{\max}$. Next we check if we can decrease $d_{t,u}^{\max}$ further by considering the minimal number of DTs per week. Finally, we set $d_{t,u}^{\max}$ to the closest working day smaller than or equal to $d_{t,u}^{\max}$.

Note that this preprocessing could be strengthened by explicitly considering the resource availabilities. However, it can be expected that w.r.t. our benchmark instances such an extension has only limited effect.

Preprocessing of Global Resource Availability Changes

The next preprocessing technique is based on the observation that due to the structure of the DTs some resources cannot be used close to global changes of resource availabilities. In such cases we are allowed to prune the availability of the resources.

More specifically, we consider here unavailability periods that are present at the same time and day over all resources. We denote such global unavailability periods by $\overline{\mathcal{W}}_d = \bigcup_{w=1,\ldots,\omega_d} \overline{\mathcal{W}}_{d,w}$ with $\overline{\mathcal{W}}_{d,w} = [\overline{\mathcal{W}}_{d,w}^{\text{start}}, \overline{\mathcal{W}}_{d,w}^{\text{end}}) \subset \widetilde{\mathcal{W}}_d$, $w = 1, \ldots, \omega_d$ during which the availability of all resources is interrupted. In addition, let $p_{r,d}^{\text{rampup}}$ be a lower bound on the earliest time a resource $r \in R$ can be used on day $d \in D'$ if all resources become available at the same time calculated as

$$p_{r,d}^{\text{rampup}} = \min_{t \in T,\, u \in U_t : r \in Q_{t,u},\, d_{t,u}^{\min} \le d \le d_{t,u}^{\max}} P_{t,u,r}^{\text{start}}. \tag{3.46}$$

Analogously, let $p_{r,d}^{\text{winddown}}$ be the minimum offset between the latest use of resource r and the end of the DT considering all DTs that might be scheduled on day d computed as

$$p_{r,d}^{\text{winddown}} = \min_{t \in T,\, u \in U_t : r \in Q_{t,u},\, d_{t,u}^{\min} \le d \le d_{t,u}^{\max}} (p_{t,u} - P_{t,u,r}^{\text{end}}). \tag{3.47}$$

We derive the following unavailability periods from the global unavailability periods:

$$[\overline{\mathcal{W}}_{d,w}^{\text{start}} - p_{r,d}^{\text{winddown}}, \overline{\mathcal{W}}_{d,w}^{\text{end}} + p_{r,d}^{\text{rampup}}) \qquad \forall r \in R, \forall d \in D_r^{\text{res}}, w = 1, \ldots, \omega_d. \tag{3.48}$$

Arc Consistency for Resource Availabilities

The preprocessing from above has only an effect in situations when all resources change their availability at the same time. Here, we consider a more widely applicable technique that is able to derive new unavailability periods from existing unavailability periods on other resources. This method is repeatedly applied, using the classic AC3 algorithm [76], until no new unavailability periods can be deduced.

On a given day $d \in D'$, an unavailability period for resource r' is derived from an existing unavailability period of resource r if all DTs requiring resource r' also need resource r. As

a first step, we have to determine the pairs of resources r and r' that allow a generation of such unavailability periods. To this end, we define for a considered working day $d \in D'$ the set R_d^{av} of resources that are available on that day and the set G_d^{pos} of DTs that might get scheduled, i.e.,

$$R_d^{\text{av}} = \{r \in R \mid d \in D_r^{\text{res}}\}, \tag{3.49}$$

$$G_d^{\text{pos}} = \{(t,u) \mid t \in T,\, u \in U_t,\, d_{t,u}^{\min} \leq d \leq d_{t,u}^{\max}\}. \tag{3.50}$$

The pairs of resources r and r' that allow a generation of unavailability periods on day d are given by

$$R_d^{\text{gen}} = \{(r,r') \mid r,r' \in R_d^{\text{av}},\, \nexists (t,u) \in G_d^{\text{pos}} (r' \in Q_{t,u} \wedge r \notin Q_{t,u})\}. \tag{3.51}$$

The length of a derived unavailability period obviously depends on the DTs requiring both considered resources. For working day d and any pair $(r,r') \in R_d^{\text{gen}}$ let $P_{r,r',d}^{\text{startdiff}}$ be the maximum offset between the first use of resource r and the first use of resource r' of all DTs in G_d^{pos} defined as

$$P_{r,r',d}^{\text{startdiff}} = \max_{(t,u) \in G_d^{\text{pos}} | r,r' \in Q_{t,u}} (P_{t,u,r}^{\text{start}} - P_{t,u,r'}^{\text{start}}). \tag{3.52}$$

Analogously, for working day d and any pair $(r,r') \in R_d^{\text{gen}}$ let $P_{r,r',d}^{\text{enddiff}}$ be the maximum offset between the last use of resource r and the last use resource r' of all DTs in G_d^{pos} specified as

$$P_{r,r',d}^{\text{enddiff}} = \max_{(t,u) \in G_d^{\text{pos}} | r,r' \in Q_{t,u}} (P_{t,u,r'}^{\text{end}} - P_{t,u,r}^{\text{end}}).$$

Finally, we derive for each pair $(r,r') \in R_d^{\text{gen}}$ and each unavailability period $\overline{W}_{r,d,w}$ in $\overline{W}_{r,d}$ the following unavailability period for r':

$$[\overline{W}_{r,d,w}^{\text{start}} + P_{r,r',d}^{\text{enddiff}}, \overline{W}_{r,d,w}^{\text{end}} - P_{r,r',d}^{\text{startdiff}}). \tag{3.53}$$

The unavailability period is then included into $\overline{W}_{r',d}$ if it is nonempty. Adjacent or overlapping unavailability periods are merged.

3.9.3 First Approaches

In this section we focus first on the day assignment part. To this end, we compare TWCH's day assignment phase, presented in Section 3.4.1, with an MILP approach. Afterwards, we consider also the time assignment level and evaluate next to TWCH the GRASP and the IG metaheuristic from Section 3.5. Note that we present here the results from Maschler et al. [81] where we considered DTs to be composed of activities associated with minimum and maximum time lags. As already mentioned this feature is not regarded to be relevant in our real world midterm planning application. Moreover, we used in Maschler et al. [81] another benchmark set compared to the later works

which mainly differs in that it had rather unrealistically strict constraints concerning the starting days of therapies, which made an extensive use of extended service time windows unavoidable. Despite these differences, the conclusions from the experiments are transferable to the problem definition presented in this chapter and to the improved benchmark instances used in the remaining chapter. However, be aware that the absolute objective values presented in this section are not comparable with the ones shown in the subsequent sections.

All algorithms used for the experiments discussed in this section were implemented in C++11 and compiled with G++ 4.8.4. GUROBI 6.5 was used for solving the MILP models. All experiments were carried out on a single core of an Intel Xeon E5-2630v2 processor with 2.6 GHz and about 4 GB RAM per core.

In the first series of conducted experiments we focus on the day assignment level only and assess the performance of TWCH's day assignment phase in comparison to the corresponding MILP model (3.4) to (3.16) from Section 3.3.1. Thus, resource consumption is only considered at the aggregated level and no detailed time planning is done. For each of our benchmark instances, the MILP was solved using a CPU-time limit of 2 hours, while TWCH's day assignment was applied 30 times because of its stochastic nature. Table 3.1 shows the obtained results for each instance. For TWCH average objective values $\overline{da\text{-}obj}$ are listed together with the corresponding standard deviations $\sigma(da\text{-}obj)$ and the median computation times *time*. For the MILP approach, *da-obj* indicates the objective value of the best feasible solution, *lb* the final lower bound, and *time* the CPU-time when the algorithm terminated, either with proven optimality or when the time limit had been reached.

TWCH yielded reasonable solutions for each test instance quickly, but improvement potential can also clearly be seen. The MILP approach could not find a provably optimal solution for any instance with $n_T \geq 50$. Only instances that allowed solutions with none or very little extended time could be reasonably solved via the MILP. A reason for the rather poor performance of the MILP seem to be substantial symmetries in the model. Due to the poor performance of the MILP for the day assignment only, we can conclude that solving the full MILP including the detailed time planning unfortunately is in practice impossible for any instance of realistic size.

In the next series of experiments we consider the complete TWCH as well as the GRASP and the IG from Section 3.5. We call the latter from here on IG-LI, where LI stands for the applied local improvement method. In a preliminary study we determined 20 CPU-minutes to be a reasonable time limit for the metaheuristics after which only minor further improvements can be expected also on the largest instances with 300 therapies. We therefore used this time limit as termination criterion in all following metaheuristic runs. The automatic parameter configuration tool irace [75] was applied for tuning IG-LI's and GRASP's strategy parameters on three instance sets with a budget of 2000 runs. The instance sets used for tuning consisted of six new training instances with 50 and 70, 100 and 150, 200 and 300 therapies. The tuned parameter settings are depicted

	TWCH's day assignment			Relaxed MILP		
Instance	$\overline{da\text{-}obj}$	$\sigma(da\text{-}obj)$	$time[s]$	$da\text{-}obj$	lb	$time[s]$
010-01	**0.220**	0.000	0.001	**0.220**	0.220	3.4
010-02	**0.160**	0.000	0.001	**0.160**	0.160	4.6
010-03	**0.160**	0.000	0.000	**0.160**	0.160	3.0
010-04	**0.180**	0.000	0.001	**0.180**	0.180	4.0
010-05	**0.180**	0.000	0.001	**0.180**	0.180	4.2
020-01	**0.400**	0.000	0.001	**0.400**	0.400	6.9
020-02	**0.450**	0.000	0.001	**0.450**	0.450	7.8
020-03	**0.460**	0.000	0.001	**0.460**	0.460	10.4
020-04	**0.320**	0.000	0.001	**0.320**	0.320	9.2
020-05	**0.320**	0.000	0.001	**0.320**	0.320	7.9
050-01	7.532	0.037	0.003	**1.770**	1.601	7200.0
050-02	4.005	0.007	0.002	**1.480**	1.443	7200.0
050-03	11.679	0.582	0.003	**2.390**	2.258	7200.0
050-04	2.597	0.000	0.003	**1.470**	1.376	7200.0
050-05	7.325	0.434	0.003	**2.317**	2.142	7200.0
070-01	37.256	1.804	0.005	**10.073**	8.362	7200.0
070-02	43.895	0.258	0.004	**14.283**	13.933	7200.0
070-03	**3.665**	0.029	0.004	4.963	2.257	7200.0
070-04	**12.187**	0.481	0.004	NA	3.934	7200.0
070-05	5.165	0.068	0.004	**2.780**	2.657	7200.0
100-01	5.110	0.000	0.005	**3.953**	3.117	7200.0
100-02	4.719	0.066	0.005	**2.970**	2.900	7200.0
100-03	8.083	0.626	0.006	**3.710**	3.592	7200.0
100-04	9.966	0.190	0.006	**4.340**	4.189	7200.0
100-05	5.713	0.162	0.006	**2.860**	2.825	7200.0
150-01	**46.317**	0.733	0.010	NA	11.528	7200.0
150-02	**30.367**	0.233	0.010	NA	7.639	7200.0
150-03	**13.787**	0.171	0.008	NA	7.176	7200.0
150-04	10.541	0.347	0.008	**5.950**	5.811	7200.0
150-05	26.764	0.499	0.009	**9.067**	8.858	7200.0
200-01	17.611	0.711	0.012	**12.167**	7.500	7200.0
200-02	**53.440**	0.649	0.012	NA	11.068	7200.0
200-03	70.021	1.166	0.013	**13.940**	13.640	7200.0
200-04	**89.349**	2.343	0.014	NA	13.035	7200.0
200-05	27.785	0.166	0.013	**11.253**	10.883	7200.0
300-01	**56.725**	1.122	0.020	NA	0.000	7200.0
300-02	**68.653**	1.221	0.020	NA	17.455	7200.0
300-03	**60.787**	0.646	0.019	NA	19.660	7200.0
300-04	10.645	0.127	0.018	**8.717**	7.631	7200.0
300-05	**69.533**	0.606	0.020	NA	16.850	7200.0

Table 3.1: Results of TWCH's day assignment and the relaxed MILP that considers only day assignments.

Instance size	GRASP			IG-LI		
n_T	$\beta^{\text{gr-rand}}$	$n^{\text{rta-noimp}}$	$k^{\text{rta-rand}}$	$\beta^{\text{ig-dest}}$	$n^{\text{rta-noimp}}$	$k^{\text{rta-rand}}$
10, 20, 50, 70	0.310	$\lfloor 1.25 \cdot \lvert G_d \rvert \rfloor$	2	0.095	$\lfloor 1.85 \cdot \lvert G_d \rvert \rfloor$	2
100, 150	0.155	$\lfloor 1.19 \cdot \lvert G_d \rvert \rfloor$	2	0.090	$\lfloor 1.50 \cdot \lvert G_d \rvert \rfloor$	2
200, 300	0.090	$\lfloor 1.28 \cdot \lvert G_d \rvert \rfloor$	2	0.110	$\lfloor 1.60 \cdot \lvert G_d \rvert \rfloor$	2

Table 3.2: Parameter settings for GRASP and IG-LI.

in Table 3.2. Note that we apply the settings for instances with 50 and 70 therapies also on the instances with 10 and 20 therapies.

Table 3.3 shows for each of the approaches and each of the benchmark instances the average final objective values $\overline{obj}$ and corresponding standard deviations $\sigma(obj)$ over 30 runs. For TWCH median computation times are also listed. For the benchmark instances with 10 and 20 therapies all three approaches always yielded the same objective values, which also coincide with the objective values from Table 3.1. This implies that in these cases the time assignment can be done without additional extended time, the instances are thus relatively easy. Since the MILP has shown that these objectives are optimal for the day assignment, it allows to conclude that these instances could also be solved optimally when additionally considering the time assignment. For all other instances, this observation does not hold anymore. It can clearly be seen that the detailed consideration of scheduling the DTs within the working days imposed significant additional costs. Both, GRASP as well as IG-LI, could find substantially better solutions than TWCH in all those cases. The clear winner, however, is IG-LI, which performed best on most instances. The major reason for its superiority seems to be the fact that on the one hand its iterations are less costly as only parts of a solution are affected by destruction and construction, and on the other hand information is in this way also kept over the iterations and further fine-tuned.

3.9.4 Enhanced Iterated Greedy

We perform in this section an experimental evaluation and comparison of the proposed enhancements of the IG introduced in Section 3.6, which we call from here on IG-DCLS, with the original IG approach from Section 3.5.2, denoted as IG-LI, and IG-LS, the variant that uses the destruction and construction phase from IG-LI combined with the local search method from Section 3.6.1. Note that there is an additional variant in which the used local search method of IG-DCLS is replaced with the local improvement operator from Section 3.5. We exclude this variant from further considerations because the local improvement operator ignores the time assignment provided by the construction phase and, hence, it is in practice equivalent to IG-LI.

For this section all algorithms have been implemented in C++11 and have been compiled with G++ 4.8.4. The experiments were performed using a single core of an Intel Xeon

	TWCH			GRASP		IG-LI	
Instance	$\overline{obj}$	$\sigma(obj)$	$time$[s]	$\overline{obj}$	$\sigma(obj)$	$\overline{obj}$	$\sigma(obj)$
010-01	**0.220**	0.000	0.001	**0.220**	0.000	**0.220**	0.000
010-02	**0.160**	0.000	0.001	**0.160**	0.000	**0.160**	0.000
010-03	**0.160**	0.000	0.001	**0.160**	0.000	**0.160**	0.000
010-04	**0.180**	0.000	0.001	**0.180**	0.000	**0.180**	0.000
010-05	**0.180**	0.000	0.001	**0.180**	0.000	**0.180**	0.000
020-01	**0.400**	0.000	0.002	**0.400**	0.000	**0.400**	0.000
020-02	**0.450**	0.000	0.002	**0.450**	0.000	**0.450**	0.000
020-03	**0.460**	0.000	0.003	**0.460**	0.000	**0.460**	0.000
020-04	**0.320**	0.000	0.002	**0.320**	0.000	**0.320**	0.000
020-05	**0.320**	0.000	0.002	**0.320**	0.000	**0.320**	0.000
050-01	159.842	0.248	0.009	120.234	0.966	**105.837**	4.452
050-02	128.793	0.320	0.007	88.034	0.921	**80.724**	1.908
050-03	161.722	2.831	0.010	117.866	1.380	**93.362**	3.907
050-04	161.795	0.221	0.008	**114.689**	1.421	122.618	2.596
050-05	177.635	0.748	0.009	131.223	0.874	**113.455**	6.263
070-01	304.241	4.370	0.012	232.000	2.282	**192.295**	3.092
070-02	278.451	2.322	0.012	211.597	2.993	**169.409**	5.157
070-03	165.936	3.618	0.011	**118.942**	1.941	121.089	4.439
070-04	194.032	3.012	0.011	148.013	1.470	**116.831**	3.644
070-05	162.713	4.288	0.010	118.022	1.985	**107.238**	4.433
100-01	183.740	1.799	0.025	**138.086**	2.319	149.066	3.388
100-02	136.303	3.488	0.028	**105.593**	0.910	106.684	2.306
100-03	245.927	4.125	0.030	185.778	1.452	**185.629**	4.320
100-04	162.602	1.583	0.030	133.788	1.505	**122.859**	2.448
100-05	247.242	4.015	0.028	179.523	2.051	**177.468**	3.556
150-01	320.521	5.625	0.049	265.495	1.779	**186.702**	3.610
150-02	372.983	4.612	0.047	300.542	2.412	**252.423**	5.615
150-03	273.096	6.973	0.041	207.086	2.536	**195.565**	4.880
150-04	182.204	4.230	0.040	131.184	2.602	**126.098**	4.576
150-05	263.687	5.103	0.045	210.231	2.104	**168.895**	3.903
200-01	340.069	7.659	0.057	255.235	2.926	**233.247**	5.986
200-02	439.731	5.956	0.058	350.984	3.355	**292.811**	6.179
200-03	487.131	4.096	0.066	409.564	1.389	**335.429**	4.576
200-04	548.790	6.364	0.066	457.902	3.994	**352.461**	8.034
200-05	317.170	2.407	0.060	263.558	1.708	**230.248**	3.667
300-01	708.705	11.009	0.098	565.907	3.573	**512.875**	4.269
300-02	727.669	13.390	0.099	579.483	3.602	**519.220**	6.955
300-03	706.027	10.762	0.098	539.983	3.464	**521.847**	6.672
300-04	527.563	7.071	0.096	**370.891**	2.556	375.673	4.558
300-05	689.882	10.095	0.099	566.615	5.095	**509.551**	4.808

Table 3.3: Average results of TWCH, GRASP and IG-LI over 30 runs.

Instance size	IG-LI			IG-LS	IG-DCLS
n_T	$\beta^{\text{ig-dest}}$	$n^{\text{rta-noimp}}$	$k^{\text{rta-rand}}$	$\beta^{\text{ig-dest}}$	$\beta^{\text{ig-dest}}$
50	0.2	$\lfloor 2.25 \cdot \lvert G_d \rvert \rfloor$	2	0.092	0.068
70	0.08	$\lfloor 1.50 \cdot \lvert G_d \rvert \rfloor$	2	0.15	0.05
100	0.055	$\lfloor 1.86 \cdot \lvert G_d \rvert \rfloor$	2	0.06	0.039
150	0.08	$\lfloor 1.70 \cdot \lvert G_d \rvert \rfloor$	2	0.106	0.026
200	0.069	$\lfloor 1.34 \cdot \lvert G_d \rvert \rfloor$	2	0.056	0.022
300	0.149	$\lfloor 2.43 \cdot \lvert G_d \rvert \rfloor$	2	0.189	0.023

Table 3.4: Parameter settings for IG-LI, IG-LS, and IG-DCLS determined by irace.

E5540 processor with 2.53 GHz. In a preliminary study we observed for the local search method that a next improvement strategy converges, in general, significantly faster than a best improvement strategy, while yielding similarly good solutions. Moreover, we tested the impact of randomizing the order in which the local search examines the neighboring solutions. It turned out that this randomization yields small improvements on almost all instances that are, however, still in the magnitude of the standard deviation. From a theoretical point of view, the randomization of the order of the considered moves removes a bias towards exchanges at the beginning of days. Therefore, we applied this randomization in the following experiments. Moreover, empirical investigations have shown that the restrictions of the neighborhood exclude promising moves only in rare cases. We adopt the acceptance criterion and the termination condition from IG-LI: The incumbent solution is replaced by a current new solution if the latter has a smaller objective value, and the total CPU-time is limited to 20 CPU-minutes, respectively.

The metaheuristics' strategy parameters were tuned using the automatic parameter configuration tool irace [75] in version 2.1. In detail, irace was applied separately on each instance size to tune $\beta^{\text{ig-dest}}$ for IG-DCLS and IG-LS and the parameters $\beta^{\text{ig-dest}}$, $n^{\text{rta-noimp}}$, and $k^{\text{rta-rand}}$ for IG-LI. On this account, we generated for each instance size five independent instances for tuning. Moreover, each irace run had a computational budget of 1000 experiments. The resulting parameter configurations are shown in Table 3.4.

Table 3.5 depicts for IG-LI, IG-LS, and IG-DCLS averages of the final objective values $\overline{obj}$ and the corresponding standard deviation $\sigma(obj)$ over 30 runs for each of the 30 benchmark instances. We start by comparing IG-LI with IG-LS. The average objective values of IG-LI are on 17 benchmark instances smaller compared to IG-LS. On most instances the absolute differences between the objective values are still in the range of the standard deviation. For this reason we applied a Wilcoxon rank sum test with a significance level of 95% for each instance. It turned out that IG-LI performed significantly better compared to IG-LS on all instances with 70 and 300 therapies and on two instances with 50 and 150 therapies, respectively. IG-LS has significantly better results on all instances with 100 therapies, on one with 50 and 150 therapies, respectively, and on two instances with 200 therapies. Thus, although IG-LI shows better results than IG-LS on

Instance	IG-LI		IG-LS		IG-DCLS	
	$\overline{obj}$	$\sigma(obj)$	$\overline{obj}$	$\sigma(obj)$	$\overline{obj}$	$\sigma(obj)$
050-01	7.512	1.179	8.945	1.953	**4.432**	1.338
050-02	43.618	2.297	50.768	4.082	**40.609**	4.003
050-03	47.355	3.373	49.522	6.438	**40.100**	4.523
050-04	22.263	1.520	**21.224**	1.828	22.536	3.136
050-05	46.567	2.978	**45.528**	2.631	50.105	5.589
070-01	26.563	3.814	33.289	2.998	**19.442**	3.187
070-02	38.545	3.958	40.184	3.390	**36.304**	5.476
070-03	**58.569**	3.720	63.721	2.517	66.849	3.316
070-04	10.615	2.566	12.764	2.655	**9.144**	2.042
070-05	44.101	2.815	46.606	2.236	**42.100**	4.075
100-01	20.210	1.477	17.861	1.328	**10.840**	0.858
100-02	36.819	3.368	30.992	3.035	**14.282**	1.993
100-03	12.045	1.321	10.363	1.039	**8.143**	0.376
100-04	21.236	2.374	19.848	2.308	**12.512**	1.333
100-05	33.870	2.126	27.253	1.924	**11.746**	1.610
150-01	25.287	2.038	27.731	2.306	**18.188**	1.662
150-02	102.486	3.085	77.613	4.166	**46.879**	3.156
150-03	36.246	2.972	41.840	3.398	**29.723**	2.955
150-04	25.392	2.803	26.329	2.994	**17.656**	1.864
150-05	34.176	2.026	34.139	2.752	**22.096**	3.504
200-01	62.931	4.031	54.072	5.134	**46.174**	4.046
200-02	62.335	4.685	60.310	3.283	**34.189**	2.393
200-03	35.798	3.109	36.137	3.163	**28.734**	2.829
200-04	51.805	3.876	49.700	3.302	**28.990**	2.581
200-05	40.434	2.669	40.231	3.207	**28.621**	2.161
300-01	23.368	1.905	26.646	2.240	**23.312**	2.965
300-02	93.680	3.613	99.561	4.988	**56.730**	4.381
300-03	**36.127**	3.920	40.530	3.472	39.223	4.111
300-04	106.270	4.870	114.920	5.685	**103.399**	6.813
300-05	20.208	1.460	21.607	1.319	**17.484**	1.750

Table 3.5: Average objective values $\overline{obj}$ of 30 runs and corresponding standard deviations $\sigma(obj)$ for IG-LI, IG-LS, and IG-DCLS.

slightly more instances, no significant overall performance difference can be observed among these two approaches. This indicates that exchanging just the local improvement operator of the IG-LI with the local search component described in Section 3.6.1 does not yield a substantial improvement.

Most importantly, however, Table 3.5 clearly shows that IG-DCLS dominates the two other metaheuristics, and provides the best average objective values on 26 out of 30 benchmark instances. According to a Wilcoxon rank sum tests with a significance level of 95%, IG-DCLS is better than IG-LI on 25 instances and better than IG-LS on 26 instances. Moreover, on 16 instances the average objective values of IG-DCLS are 25% smaller compared to those of IG-LI, and on three instances the average objective values are halved compared to the ones from IG-LI. The main reason for this performance improvement is the interplay between IG-DCLS's construction phase and local search procedure. On the one hand, the local search operator is, in general, able to provide better results than IG-LI's local improvement operator. However, encoding, decoding, and evaluating the solution is computationally demanding and, hence, converging to a local optimum is time consuming, especially on strongly perturbed solutions. On the other hand, IG-DCLS's construction phase is designed in such a way that large parts of the sequence of the DTs are preserved while introducing the removed DTs in a sensible but randomized way. Starting with a solution close to a local optimum w.r.t. the DT exchange neighborhood allows to reduce the time spent in the local search procedure and, consequently, increases the total number of iterations.

The impact of IG-DCLS's construction phase is also implicitly reflected in the parameter configurations determined by irace. IG-LS' local search starts each iteration with a new time assignment produced by TWCH's time assignment from scratch. Due to the many steps required to reach a locally optimal solution it makes sense to use a rater high destruction rate to cover a larger part of the search space within the time limit. With the new destruction and construction phase, however, the situation changes, because now there is an immediate correspondence between the destruction rate and the time required to reach a locally optimal solution and with it the number of total iterations. Here, the destruction is able to either focus or broaden the exploration of the search space.

3.9.5 Time Estimation for Scheduling Sets of Treatments

In this section we study the performance impact of applying the presented time estimation within our improved IG metaheuristic, the IG-DCLS described in Section 3.6. Moreover, we determine the accuracy of the surrogate functions on a final solution. We use the same experimental setup as in the previous section.

Table 3.6 compares the performance between the IG-DCLS, as presented in Section 3.6, with the variant of the IG-DCLS where in the day assignment the time required from the beam and room resources phase is estimated by (3.34), (3.35), and (3.36). Both algorithms use as termination criterion a time limit of 20 CPU-minutes and are executed on each of the benchmark instances for 30 times. Table 3.6 shows the mean objective

	IG-DCLS				IG-DCLS with time estimation			
Instance	$\overline{obj}$	$\sigma(obj)$	$\overline{ext}[h]$	$\sigma(ext)$	$\overline{obj}$	$\sigma(obj)$	$\overline{ext}[h]$	$\sigma(ext)$
050-01	5.139	1.526	2.533	1.526	**2.468**	0.085	**0.000**	0.085
050-02	43.525	4.096	38.158	4.096	**33.233**	2.071	**27.950**	2.071
050-03	36.069	5.439	30.458	5.439	**30.364**	4.479	**25.183**	4.479
050-04	24.076	2.613	19.758	2.613	**20.261**	1.860	**15.275**	1.860
050-05	51.128	4.351	47.167	4.351	**42.625**	3.055	**37.233**	3.055
070-01	19.612	4.385	15.183	4.385	**10.823**	1.700	**5.558**	1.700
070-02	40.407	6.780	33.800	6.780	**33.192**	3.990	**27.850**	3.990
070-03	69.194	3.983	62.742	3.983	**60.801**	4.285	**53.042**	4.285
070-04	9.784	2.485	4.050	2.485	**5.662**	0.385	**0.000**	0.385
070-05	46.702	5.195	40.917	5.195	**43.069**	2.994	**36.250**	2.994
100-01	11.558	0.976	5.483	0.976	**9.046**	0.468	**2.633**	0.468
100-02	15.508	2.372	8.158	2.372	**9.737**	0.774	**1.933**	0.774
100-03	8.488	0.497	3.192	0.497	**6.435**	0.161	**0.883**	0.161
100-04	14.257	1.566	7.117	1.566	**8.884**	0.473	**1.125**	0.473
100-05	13.826	1.755	7.817	1.755	**6.823**	0.235	**0.000**	0.235
150-01	18.916	1.760	7.417	1.760	**14.068**	0.387	**1.733**	0.387
150-02	52.166	4.722	39.500	4.722	**43.950**	3.475	**30.092**	3.475
150-03	32.886	3.757	20.233	3.757	**32.740**	3.390	**20.142**	3.390
150-04	18.620	1.659	6.875	1.659	**12.395**	0.466	**0.033**	0.466
150-05	24.286	3.833	15.483	3.833	**10.628**	0.631	**0.917**	0.631
200-01	48.102	3.935	34.000	3.935	**35.945**	5.275	**17.225**	5.275
200-02	38.085	2.824	21.533	2.824	**35.206**	3.855	**16.442**	3.855
200-03	31.158	3.574	13.075	3.574	**20.454**	0.895	**1.108**	0.895
200-04	30.913	2.576	16.800	2.576	**18.860**	1.324	**2.075**	1.324
200-05	29.846	2.384	17.092	2.384	**19.876**	2.550	**5.600**	2.550
300-01	23.654	2.739	12.067	2.739	**16.429**	1.459	**4.000**	1.459
300-02	61.320	5.063	41.200	5.063	**52.510**	5.979	**27.608**	5.979
300-03	41.415	4.254	25.392	4.254	**23.707**	2.900	**6.350**	2.900
300-04	108.118	7.221	85.608	7.221	**77.244**	4.501	**50.367**	4.501
300-05	18.684	1.634	6.800	1.634	**13.219**	0.586	**0.833**	0.586

Table 3.6: Average objective values $\overline{obj}$ and average use of extended service periods in hours $\overline{ext}[h]$ of 30 runs with a time limit of 20 CPU-minutes and corresponding standard deviations $\sigma(obj)$ and $\sigma(ext)$ for IG-DCLS and IG-DCLS with time estimation.

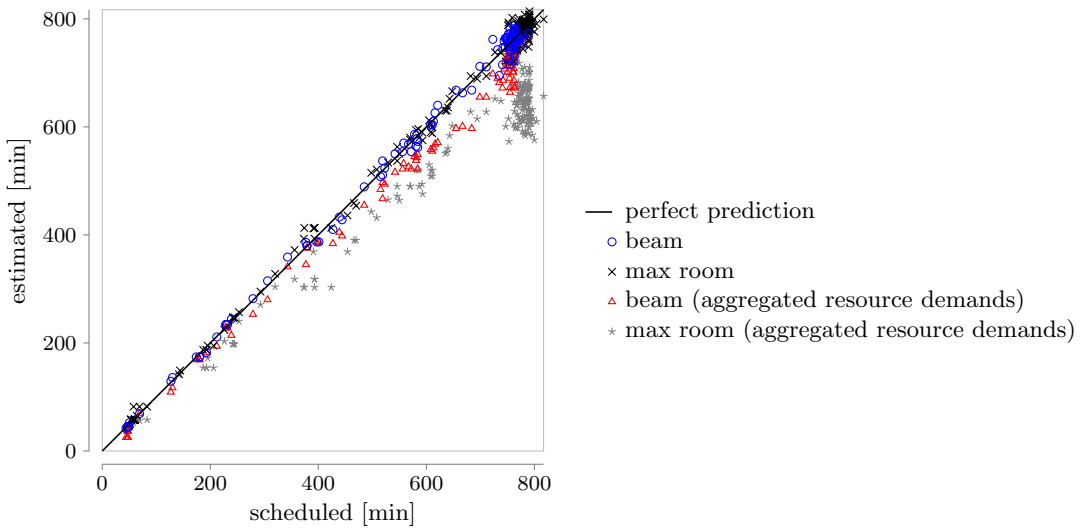


Figure 3.7: Prediction quality of the improved time estimation for the beam resource and the most used room resource in comparison with the trivial lower bound given by aggregated resource demands.

values $\overline{obj}$ and the median use of extended service periods $ext[h]$ in hours with the corresponding standard deviations $\sigma(obj)$ and $\sigma(ext)$ of finally obtained solutions. The results indicate that the application of the presented estimation considerably reduces the used extended service periods over all benchmark instances. The surrogate functions are, however, not necessarily a lower bound. Thus, we might occasionally overestimate the required time for the bottleneck resources yielding underutilized days. The consequence in general is that the finishing day of therapies are delayed, which is penalized with the second term of our objective function. This raises the question whether the trade-off is indeed beneficial w.r.t. objective function (3.1) using the usual weights. This is indeed the case, since the IG-DCLS with the presented time estimation performs on all benchmark instances significantly better than the one without according to a Wilcoxon rank sum test with a significance level of 95%.

The performance improvements can be explained by the increased accuracy of $\widehat{\lambda}_{r,d}$. To this end, we consider the final solution of a randomly selected run for a benchmark instance with 300 therapies. Figure 3.7 compares for all used working days our improved time estimation and the trivial lower bound given by aggregated resource demands with the actual required time. Using the trivial lower bound underestimates on average the required time from the beam and the most used room by 27.7 and 101.6 minutes (i.e., by 6.5% and 15.3%) with a standard deviation of 20.2 and 57.2, respectively. The presented estimation is on average off by 9.17 minutes for the beam and by 10.9 minutes for the most used room (i.e., by 2% and 2.4%) with a standard deviation of 8.6 and 9.6, respectively.

3.9.6 Iterated Greedy for Limiting Starting Time Variations

In this section we perform an experimental evaluation of the proposed IG approach from Section 3.8 for the extended problem formulation, which we call from here on IG2. As reference method we use a variant of the IG from Section 3.5.2, denoted as IG-LI, that is revised for considering the extended objective function (3.3). In the experiments we start with IG-LI and replace its components step-by-step with the ones from IG2. The aim is to investigate on the properties and impacts of the individual proposed enhancements and to demonstrate that their combination yields indeed an improved metaheuristic.

The algorithms for the following experiments have been implemented in C++14 and compiled with G++ 6.3.0, and all experiments were carried out using a single core of an Intel Xeon E5-2640 v4 CPU with 2.40GHz. The LP models for determining the nominal starting times have been solved with Gurobi 7.5. We again adopt the acceptance criterion and the termination condition from IG-LI: The incumbent solution is replaced by a current new solution iff the latter has a smaller objective value, and the total CPU-time is limited to 20 minutes, respectively.

To use IG-LI as reference algorithm we have to provide an extension that determines in addition the nominal starting times. This can be done by solving the LP model presented in Section 3.8.2 for the initial solution provided by TWCH and at the end of each construction phase. Moreover, like in the local search procedure presented in Section 3.8.2 we repeatedly apply the local improvement method followed by a recalculation of the nominal starting times until no improvements can be found. As already stated in Section 3.8.1, it is not straightforward how to extend the time assignment part of TWCH s.t. it explicitly respects the additional objective terms. However, due to the behavior of the time assignment to prioritize DTs with certain properties we can observe quite frequently that DTs belonging to the same therapy are scheduled roughly at the same time anyway.

To show that the IG approach presented in this work indeed enhances IG-LI we will gradually exchange components of IG-LI with the ones presented here. In this way we assess the properties of the individual components and their interplay. We start by exchanging in IG-LI the local improvement component with the local search procedure from Section 3.8.2. We call the resulting algorithm IG-LS. Afterwards we interchange in addition the destruction and construction phase, which is named IG-DCLS. Finally, we swap in IG-DCLS the construction heuristic for the initial solution and obtain the final IG2. Note that we receive all meaningful variants between IG-LI and IG2 by exchanging the metaheuristic's components in this order. Starting with applying the proposed local search method instead of the local improvement makes sense since it is performed last within each iteration. In contrast, starting with replacing the destruction and construction component with the one from Section 3.8.3 yields a conceptually flawed variant. While the construction phase aims at keeping relative timing characteristics of not removed DTs, the local improvement operator ignores the time assignment provided by the construction phase. Hence, either the construction phase or the local improvement becomes dominant

and ignores the influence of the other. Exchanging the construction heuristic for the initial solution after the construction phase is also reasonable for the same argument: In an IG variant using the newly assembled construction heuristic combined with the construction phase of IG-LI the first successful iteration would completely revoke the time assignment of the initial solution.

In a preliminary study we experimented with several neighborhoods in place of the exchange neighborhood within the IG2. We considered two variants of insertion neighborhoods. The moves of the first consist of removing and reinserting a DT in another position in the sequence obtained by decoding a considered day. Besides of being larger than the exchange neighborhood, insertion moves are less likely to improve the solution due to the interleaving of DTs w.r.t. the used rooms in good solutions. The second considered insertion neighborhood removes and reinserts DTs but changes only the positions of the DTs requiring the same room resource. The exchange neighborhood performs better for two reasons. One the one hand, DTs have in general different timing characteristics and resource requirements. Thus, keeping the positions of the DTs requiring other room resources fixed can cause disruptions in the schedule as well. On the other hand, if we only modify the positions of the DTs requiring the same room, then the starting times of the affected DTs change typically to larger degree. This may cause frequently an increase on the objective terms regarding the nominal starting times. Moreover, we also tried a neighborhood based on inversions of parts of the DT sequence on a considered day. Clearly, inversions of sequences of two or three subsequent DTs are already covered by exchanges. Inversion of very long sequences of DTs are very likely to accumulate large costs from the deviations of nominal starting times. Therefore, we considered inverting sequences up to a given maximal length. Although being smaller than the exchange neighborhood and, hence, allowing more iterations of the IG, it turned out that the overall performance is weaker compared to the presented approach. Most likely many improving exchanges of more distant DTs w.r.t. the job sequence cannot be replicated by inversions. Finally, we observed for the presented local search method that a next improvement strategy converges, in general, significantly faster than a best improvement strategy, while yielding similarly good solutions.

The enhancements of our original IG from Section 3.6 focused on exchanging the local search and the construction operator. The experiments conducted in Section 3.9.4 indicate that exchanging just the local improvement operator of the IG-LI with the local search component does not yield a substantial improvement. If, however, both the local search and the new construction phase are used together then the resulting approach clearly dominates IG-LI. The main reason for this performance improvement is the interplay between the construction phase and the local search procedure. On the one hand, the local search operator is, in general, able to provide better results than IG-LI's local improvement operator. However, encoding, decoding, and evaluating the solution is computationally demanding and, hence, converging to a local optimum is time-consuming, especially on strongly perturbed solutions. On the other hand, the construction phase is designed in such a way that large parts of the sequence of the DTs are preserved while

Instance size	IG-LI			IG-DCLS		IG2	
n_T	$\mathcal{U}^{\text{ig-dest}}$	$n^{\text{rta-noimp}}$	$k^{\text{rta-rand}}$	$\mathcal{U}^{\text{ig-dest}}$	k^{frb4}	$\mathcal{U}^{\text{ig-dest}}$	k^{frb4}
50	$\mathcal{U}\{2, 18\}$	1.881	2	$\mathcal{U}\{2, 2\}$	67	$\mathcal{U}\{2, 4\}$	24
70	$\mathcal{U}\{6, 7\}$	2.300	2	$\mathcal{U}\{2, 3\}$	46	$\mathcal{U}\{2, 9\}$	16
100	$\mathcal{U}\{6, 10\}$	0.513	7	$\mathcal{U}\{3, 3\}$	2	$\mathcal{U}\{3, 3\}$	1
150	$\mathcal{U}\{3, 8\}$	0.660	9	$\mathcal{U}\{2, 2\}$	12	$\mathcal{U}\{2, 2\}$	12
200	$\mathcal{U}\{6, 9\}$	0.598	6	$\mathcal{U}\{2, 3\}$	17	$\mathcal{U}\{2, 2\}$	14
300	$\mathcal{U}\{29, 45\}$	2.420	2	$\mathcal{U}\{2, 5\}$	25	$\mathcal{U}\{2, 2\}$	44

Table 3.7: Parameter settings for IG-LI, IG-DCLS, and IG2 determined by irace.

introducing the removed DTs in a sensible but randomized way. Starting with a solution close to a local optimum w.r.t. the DT exchange neighborhood allows to reduce the time spent in the local search procedure and, consequently, increases the total number of iterations. Although these experiments have been conducted on the original version of the PTPSP and with a slightly simplified algorithm, these results can be replicated also for the extended problem formulation. Therefore, we consider here only IG-DCLS further.

The metaheuristics' strategy parameters were tuned using the automatic parameter configuration tool irace [75] in version 2.4. In detail, irace was applied in two rounds for each instance size separately. To this end, we used an independent set of instances designated for tuning and a computational budget of 2000 experiments for each application of irace. In the first round we used irace with a larger amount of parameters for determining an effective design of the algorithm. During this configuration phase we employed FRB4$_k$ instead of FRB3 within the construction heuristic for the initial solution. The values obtained for parameter k for the initial solution corresponded with the number of DTs assigned to a fully utilized day, i.e., $k \approx n$, and thus FRB4$_k$ degenerates to FRB3. This suggests that the comprehensiveness and the implied computational costs of FRB3 are justified and beneficial for the overall approach. For the local search method we tested randomizing the order in which the local search examines the neighboring solutions. It turned out that this randomization is favorable over considering moves in the order of the starting times of respective DTs. From a theoretical point of view, the randomization of the order of the considered moves removes a bias towards exchanges at the beginning of days. Moreover, we raced whether to activate the accelerations in the local search that prematurely terminate the evaluation of a neighbor. While the technique described in Section 3.8.2 has been activated on all instance sizes, a complementary method considering increased variations to the nominal starting times has been rejected. In the second round of the algorithm configuration we kept the assessed design choices fixed to focus on the central parameters: $\mathcal{U}^{\text{ig-dest}}$, $n^{\text{rta-noimp}}$, $k^{\text{rta-rand}}$ for IG-LI and $\mathcal{U}^{\text{ig-dest}}$, k^{frb4} for IG-DCLS and IG2. As described in Section 3.8.3, $\mathcal{U}^{\text{ig-dest}}$ specifies a

discrete uniform distribution $\mathcal{U}(a, a+b)$ from which in each iteration a random number of therapies to destroy is sampled. The parameter configuration determined values for a and b, where $a \in \{1, \ldots, 40\}$ and $b \in \{0, \ldots, 40\}$. For $n^{\text{rta-noimp}}$ and $k^{\text{rta-rand}}$ we used the value ranges $[0.5, 2.5]$ and $\{1, \ldots, 10\}$, respectively. We denote with k^{frb4} the parameter k of the FRB4$_k$ algorithm used in the construction phase and consider values from $\{0, \ldots, 70\}$. The resulting parameter configurations are shown in Table 3.7.

Table 3.8 depicts for IG-LI, IG-DCLS, and IG2 averages of the final objective values $\overline{obj}$ and the corresponding standard deviation $\sigma(obj)$ over 30 runs for each of the 30 benchmark instances. Moreover, Table 3.8 gives the probability values obtained by an application of an one-tailed Wilcoxon rank sum test on the objective values of two methods at a time. We start by comparing IG-LI with IG-DCLS. The average objective values of IG-DCLS are on 29 out of 30 benchmark instances smaller than those obtained by IG-LI. The Wilcoxon rank sum test indicated with a confidence level of 95% that IG-DCLS yields significantly better solutions than IG-LI on 29 benchmark instances, the only exception is instance 050-05. In fact, on 21 instances the average objective values are halved compared with the ones from IG-LI, and on eight instances the average objective values of IG-DCLS are even 75% smaller compared with those of IG-LI. These results confirm the outcome of the experiments conducted for the original variant of the PTPSP in Section 3.9.4, which showed the advantage of applying the new construction and local search components. In fact, the superiority of IG-DCLS over IG-LS is here even more predominant. This can be explained by the fact that we use here FRB4$_k$ instead of NEH (i.e., FRB4$_0$). Moreover, the new construction operator as well as the local search operator are able to handle the more complicated objective much better than their counterparts within IG-LI.

We continue by taking our main approach into consideration. Table 3.8 clearly shows that IG2 outperforms the two other metaheuristics, and provides the best average objective values on 26 out of 30 benchmark instances. As before we applied a Wilcoxon rank sum test on the objective values with a confidence level of 95% for each instance to compare IG2 with IG-LI and IG2 with IG-DCLS. The former tests showed that IG2 performs significantly better on all benchmark instances. The latter series of statistical tests evinced that the IG2 outperforms IG-DCLS on 18 benchmark instances significantly, while the observed better average objective values of IG-DCLS have been two times significant. Furthermore, there is clear tendency that IG2 significantly performs better than IG-DCLS on larger benchmark instances. This indicates that exchanging the construction heuristic for the initial solution with the one described in Section 3.8.1 becomes of greater importance with the increasing size of the instances. The reason for this is that with larger instance sizes also the computational cost for each iteration of the IG increases and, consequently, the number of executed iterations decreases. Hence, quality of the initial solution becomes with larger instance sizes more and more important. This argument is further supported by the fact that the construction heuristic presented here requires substantially more of the total time budget of 20 CPU minutes than the application of TWCH: The average computation time for the construction heuristic based

	IG-LI (1)		IG-DCLS (2)		IG2 (3)		Wilcoxon rank sum test		
Instance	$\overline{obj}$	$\sigma(obj)$	$\overline{obj}$	$\sigma(obj)$	$\overline{obj}$	$\sigma(obj)$	$p_{1\leq 2}$	$p_{1\leq 3}$	$p_{2\leq 3}$
050-01	8.570	1.323	2.784	0.562	**2.608**	0.215	**0.000**	**0.000**	**0.027**
050-02	52.830	2.228	40.005	3.006	**39.812**	2.398	**0.000**	**0.000**	0.339
050-03	47.600	2.022	**37.281**	3.821	37.660	3.252	**0.000**	**0.000**	0.733
050-04	29.561	1.564	26.419	3.743	**25.795**	3.152	**0.000**	**0.000**	0.261
050-05	53.363	1.996	54.089	2.761	**49.848**	2.725	0.893	**0.000**	**0.000**
070-01	28.600	2.826	**13.242**	1.742	14.015	1.787	**0.000**	**0.000**	0.945
070-02	48.346	2.861	44.871	5.588	**41.540**	4.556	**0.003**	**0.000**	**0.007**
070-03	74.141	3.957	70.376	4.897	**70.227**	5.833	**0.000**	**0.001**	0.524
070-04	13.826	1.794	7.710	1.907	**7.313**	1.556	**0.000**	**0.000**	0.282
070-05	56.838	4.016	49.328	4.593	**49.316**	3.518	**0.000**	**0.000**	0.371
100-01	81.405	5.025	18.383	2.296	**17.840**	2.524	**0.000**	**0.000**	0.075
100-02	108.194	5.006	24.366	2.041	**23.552**	2.179	**0.000**	**0.000**	0.099
100-03	53.978	3.251	15.564	1.592	**15.306**	1.457	**0.000**	**0.000**	0.275
100-04	88.931	5.892	**19.703**	1.478	20.865	1.694	**0.000**	**0.000**	0.992
100-05	77.063	3.895	**14.762**	1.256	16.089	1.235	**0.000**	**0.000**	1.000
150-01	128.564	7.308	40.590	5.421	**35.605**	5.399	**0.000**	**0.000**	**0.001**
150-02	273.450	10.896	80.328	4.877	**72.008**	5.785	**0.000**	**0.000**	**0.000**
150-03	127.236	5.736	72.212	6.622	**62.368**	5.999	**0.000**	**0.000**	**0.000**
150-04	116.294	5.290	32.255	2.126	**27.787**	1.744	**0.000**	**0.000**	**0.000**
150-05	96.049	6.785	22.889	2.019	**20.093**	1.619	**0.000**	**0.000**	**0.000**
200-01	194.440	9.204	76.943	5.658	**61.244**	5.783	**0.000**	**0.000**	**0.000**
200-02	223.821	7.241	88.747	5.988	**78.073**	6.780	**0.000**	**0.000**	**0.000**
200-03	165.977	8.092	57.843	4.214	**49.197**	3.277	**0.000**	**0.000**	**0.000**
200-04	212.980	9.355	63.085	7.408	**51.859**	6.435	**0.000**	**0.000**	**0.000**
200-05	188.749	7.701	44.143	3.253	**33.294**	2.572	**0.000**	**0.000**	**0.000**
300-01	239.586	5.072	52.349	4.503	**35.082**	3.149	**0.000**	**0.000**	**0.000**
300-02	344.637	9.689	133.092	7.858	**128.118**	10.560	**0.000**	**0.000**	**0.028**
300-03	249.621	10.444	64.612	5.839	**48.084**	3.599	**0.000**	**0.000**	**0.000**
300-04	370.461	9.514	162.315	8.896	**115.586**	8.354	**0.000**	**0.000**	**0.000**
300-05	251.234	7.381	48.194	2.936	**35.885**	2.312	**0.000**	**0.000**	**0.000**

Table 3.8: Comparison of IG-LI, IG-DCLS, and IG2. We consider for each instance and every approach average objective values $\overline{obj}$ of 30 runs and corresponding standard deviations $\sigma(obj)$. Moreover, the p-values originating from an application of the Wilcoxon rank sum test on pairs of algorithms are given. To this end, we denote with $p_{A\leq B}$ the p-values under the null hypothesis that approach A performs better than or equal to method B. We mark for each instance the best average objective value and p-values smaller than 0.05.

	IG-LI		IG-DCLS		IG2	
Instance	$\overline{ext+fin}$	$\overline{iaw+iew}$	$\overline{ext+fin}$	$\overline{iaw+iew}$	$\overline{ext+fin}$	$\overline{iaw+iew}$
050-01	2.834	5.735	2.770	0.014	**2.602**	**0.006**
050-02	37.729	15.100	37.613	**2.392**	**36.908**	2.903
050-03	**33.642**	13.958	34.373	**2.908**	34.533	3.127
050-04	**20.834**	8.726	24.393	**2.026**	23.694	2.100
050-05	**40.899**	12.464	50.447	3.642	46.279	**3.569**
070-01	16.812	11.788	**12.934**	**0.308**	13.615	0.400
070-02	**32.592**	15.754	42.868	**2.003**	38.572	2.968
070-03	**55.566**	18.574	67.096	**3.280**	65.863	4.364
070-04	**6.325**	7.501	7.461	**0.249**	6.769	0.544
070-05	**41.663**	15.175	47.482	**1.846**	46.451	2.866
100-01	23.018	58.387	13.869	4.514	**13.422**	**4.418**
100-02	31.548	76.646	17.166	**7.199**	**16.142**	7.410
100-03	**10.263**	43.715	10.894	**4.670**	10.413	4.893
100-04	26.416	62.514	**13.340**	**6.364**	14.018	6.847
100-05	24.704	52.359	**9.398**	5.364	11.085	**5.004**
150-01	34.650	93.914	28.387	12.203	**26.102**	**9.504**
150-02	134.003	139.447	59.588	20.740	**52.431**	**19.577**
150-03	51.618	75.619	59.815	**12.396**	**48.742**	13.626
150-04	24.568	91.727	20.950	11.305	**17.758**	**10.028**
150-05	24.652	71.396	14.471	8.417	**13.901**	**6.192**
200-01	66.198	128.243	52.634	24.309	**39.416**	**21.828**
200-02	86.416	137.405	67.024	21.723	**56.513**	**21.561**
200-03	36.428	129.549	35.604	22.239	**29.719**	**19.478**
200-04	67.216	145.764	45.907	17.178	**34.686**	**17.174**
200-05	46.356	142.393	28.765	15.378	**20.421**	**12.873**
300-01	34.580	205.006	29.479	22.870	**20.496**	**14.586**
300-02	131.749	212.888	96.046	**37.046**	**90.977**	37.141
300-03	42.013	207.609	39.849	24.764	**30.603**	**17.481**
300-04	120.678	249.783	115.095	47.220	**76.253**	**39.334**
300-05	32.446	218.789	24.558	23.636	**21.373**	**14.512**

Table 3.9: The breakdown of the average objective values of 30 runs for IG-LI, IG-DCLS, and IG2 presented in Table 3.8 into the first two and the last two terms of our objective function (3.3) denoted as $\overline{ext+fin}$ and $\overline{iaw+iew}$, respectively.

on FRB3 is on the largest instances 29.8 seconds, while the computation of the initial solution for IG-LI and IG-DCLS takes on average only 0.25 seconds. On this account it becomes more evident that the combination of TWCH with FRB3 is indeed advantageous and that the large processing time is well spent.

Table 3.9 gives a more detailed breakdown of the average objective values presented in Table 3.8. To this end, we denote with $\overline{ext+fin}$ the part of the average objective values originating from the use of extended service time windows and from the delayed completion of therapies. The sum of these two objective terms correspond to the objective function used in the original variant of the PTPSP. Moreover, $\overline{iaw+iew}$ stands for the part of the average objective values that arise from the intra-week and inter-week objective terms. For a well-performing method it is clearly not sufficient to focus on only one part of the objective function while neglecting the other aspects of the problem. This is especially visible for the smaller benchmark instances. On the instances with 50 and 70 therapies IG-LI frequently provides solutions with the smallest costs w.r.t. the first two terms of objective function (3.3). This comes, however, with a much higher cost on the intra-week and inter-week objective terms compared with the other two approaches. Furthermore, IG-DCLS is on several occasions able to provide solutions with less excess on the allowed variations of the DTs' starting times. Nevertheless, IG2 outperforms the other two approaches on almost all cases due to the better balance between the objective parts. On the larger benchmark instances the superiority of IG2 over IG-LI and IG-DCLS becomes more evident. For most benchmark instances with more than 100 therapies the IG2 metaheuristic is able to provide the best results on both parts of the objective function.

3.10 Conclusions

We have seen that the midterm patient scheduling problem arising in modern cancer treatment centers applying particle therapy is particularly challenging. It involves planning on day level as well as a dependent detailed scheduling of DTs at each day. We presented a MILP model for the whole problem, but it became clear that already solving just the day planning part is practically intractable for most instances of realistic size. Therefore, we considered a therapy-wise construction heuristic based on greedy principles featuring a forward-looking mechanism to avoid too naive decisions. This heuristic is fast and provides already reasonable solutions. We further built upon this construction heuristic, proposing a first GRASP and IG metaheuristic. In our experiments on newly created benchmark instances based on properties of realistic scenarios, the IG approach yielded the best results. Its superiority over GRASP can be explained by the computationally more efficient destruction and construction: Only parts of each solution are reconsidered from scratch, and substantial information survives from one iteration to the next and is further fine-tuned. All proposed metaheuristics scale well to instances of practically relevant size.

Nevertheless, this first IG metaheuristic did not exhaust its full potential. Consequently, we focused in the following on improving the IG further. In contrast to this first IG, the proposed enhanced approach aims at preserving the order of the not removed DTs on the individual days. The resulting advantage is that more information from the incumbent solution is maintained. Compared to our previous IG and a variant of the previous IG that uses the local search procedure of our new IG metaheuristic provides significantly better results on 25 and 26 out of 30 benchmark instances, respectively. The superiority of the enhanced approach over the other two can be explained by the interplay between its construction phase and the newly applied local search technique: The local search method yields, in general, better results than applying TWCH's time assignment randomized for many times. However, due to the required encoding and decoding steps, evaluating neighbors is time-consuming. Hence, to ensure that the metaheuristic is able to perform sufficiently many iterations it is required that the neighborhood requires on average only a few steps until it reaches a local optimum. To this end, we apply in the construction phase an insertion heuristic that iteratively places the removed DTs into the permutation resulting from sorting the DTs according to the times at which they use the beam.

Afterwards, we presented a surrogate model for estimating the total times the bottleneck resources are required to optimally schedule sets of DTs. This surrogate model is applied to quickly estimate the use of extended service times at the day assignment level of the PTPSP. We evaluated the effects of the presented surrogate model within the improved IG algorithm. Results show that on all considered benchmark instances the use of extended service periods as well as the whole objective value can be significantly decreased. This can be explained by the substantial gain in accuracy of the new surrogate model and with it the better adjustment of the two levels.

The basic problem formulation allowed considering each day individually after the day assignment has been determined. In the extended version of the PTPSP starting time variations of DTs belonging to the same therapy should not exceed specified thresholds. To this end we introduced nominal starting times that serve as reference point for computing the variation within and between weeks. From a practical viewpoint this extension increases the difficulty of the problem substantially since on the one hand the calculation of the nominal starting times requires the DTs' starting times while on the other hand finding good starting times involves knowing the nominal starting times. Moreover, minimizing the variation of starting times is frequently contrary to our main objective which is to minimize the use of extended service time windows. Consequently, also the design of an effective metaheuristic is more involved as it requires to balance the different aspects of the objective function.

To tackle this extended problem formulation we build on our previously developed IG metaheuristics. The approach features a construction heuristic that combines parts of TWCH and the FRB heuristics from Rad et al. [99]. Moreover, a local search technique is applied that alternately examines an exchange neighborhood to improve the DTs starting times and solves an LP model that computes nominal starting times. Like our enhanced

IG for the basic problem formulation, the presented approach is able to preserve the order of the not removed DTs on the individual days which allows maintaining timing information sensibly.

To evaluate the performance of the proposed IG for the extended problem formulation we revised our first IG to cover the extended PTPSP variant. Afterwards, we replaced components of this reference algorithm step-by-step with the newly described ones to assess their properties. We again observed the beneficial interplay between the construction phase and the applied local search method: The local search yields, in general, good solutions but is due to the encoding and decoding steps time critical. The insertion heuristic based on $FRB4_k$ applied in the construction phase on the other hand is able to provide starting solutions for the local search s.t. on average only a few moves suffice to reach a local optimum. Moreover, although being computationally expensive, the newly presented construction heuristic for the initial solution, based on TWCH and FRB3, gives the IG a superior starting point. The resulting approach outperforms our reference method on all benchmark instances significantly.

A remaining challenge is to determine suitable parameter configurations for real world instances. Although the considered benchmark instances aim at modeling the expected situation at MedAustron, they contain assumptions and simplifications which might differ in the future practice. The main characteristic of the used benchmark instances is the number of therapies which have to be scheduled. Our experiments showed that the values of some parameters are highly dependent on the instance size. For real world instances it is likely that also other aspects which have to be considered for obtaining good parameter configurations. Hence, a next step to improve the applicability of the presented approach is to define a parameter model which determines values for the IG's strategy parameters on the basis of the observed instance characteristics.

PTPSP, as defined here, still simplifies the midterm planning part arising in practice. We consider therapies to consist only of DTs. In the general case, however, there is a treatment planning phase preceding all DTs. In this stage personalized equipment needed for irradiation is produced. Moreover, a substantial amount of planning has to be done to determine the detailed treatment strategy. Since other constraints have to be enforced for these tasks, they cannot be modeled as DTs. There are further activities (e.g., control examinations) complementing the core therapies that have to be provided once a week before or after one of the DTs. However, from a practical viewpoint these additional activities should never influence the throughput of the facility and can be sufficiently well handled in a post-processing step.

We formulated PTPSPs as a single-objective optimization problem by using a linear combination of the objective goals. A further natural next step would be to consider PTPSP as a multi-objective optimization problem. Moreover, for the staff it is preferable to avoid gaps within the workday. To address this minimizing the time a resource stays idle between subsequent activities should be considered as an additional objective goal.

The presented IG metaheuristics use an acceptance criterion that accepts a current solution if it has a better objective value. To improve our IG further it makes sense to also consider acceptance criteria that allow selecting suboptimal solutions, e.g., in a simulated annealing like fashion (see [107]). Moreover, the proposed local search procedure exchanges the DTs only within days. Neighborhoods that can alter the days on which a therapy is applied seem promising. The main challenge here is to design and restrict the neighborhoods s.t. the resulting local search method is still fast enough to allow a sufficiently large number of IG iterations.

CHAPTER 4

Prize-Collecting Job Sequencing

In the course of our investigation into the real-world patient scheduling problem discussed in Chapter 3 it became apparent that the scheduling of *daily treatments* (DTs) within individual days is an challenging problem on its own, which deserves further attention. In this chapter we, thus, concentrate on the most central aspects of the time assignment part for a single day. The introduced problem considers only two kinds of resources: a single resource shared among all jobs and several secondary resources where each is required by a subset of the jobs. These resources correspond to the beam and the treatment rooms in our particle therapy scenario. Similar to the DTs from Chapter 3, the jobs need one of the secondary resources for their entire execution while the resource required by all jobs is only used for a part of the processing time. Instead of resource availabilities, we consider here the slightly more general variant of having individual time windows for the jobs. New is the prize-collecting aspect, where we aim at finding a subset of jobs that can be feasible scheduled and maximizes the total prize associated with each of the jobs. This objective can be motivated by the fact that we have to frequently delay the days on which DTs are performed to avoid excessive use of extended time. Accordingly, the prizes associated with the jobs might reflect the urgency of performing a DT on the considered day.

We approach the problem by the means of *decision diagrams* (DDs) (see Section 2.4). They have been recognized to be a powerful tool for certain COPs. Relatively compact relaxed and restricted DDs allow to obtain dual bounds and heuristic solutions. The prize-collecting aspect has, to the best of our knowledge, not been studied before and provides new challenges and opportunities. Our first work on this topic has been presented at the *12th International Conference on the Practice and Theory of Automated Timetabling* (PATAT'18) and is published in the conference's proceedings:

> J. Maschler and G. R. Raidl. Multivalued decision diagrams for a prize-collecting sequencing problem. In *Proceedings of the 12th International Conference of the*

Practice and Theory of Automated Timetabling, pages 375–397, Vienna, Austria, 2018.

The primary aim of this paper is to study different methods found in the literature for creating relaxed DDs for our prize-collecting job sequencing problem. To this end, we adopt and extend the two main DD compilation approaches found in the literature: top down construction and incremental refinement. In a series of computational experiments these methods are compared.

A substantially extended version of this work has been submitted to the special issue *The Practice and Theory of Automated Timetabling* of the journal *Annals of Operations Research*:

J. Maschler and G. R. Raidl. Multivalued decision diagrams for prize-collecting job sequencing with one common and multiple secondary resources. *Annals of Operations Research*, submitted.

New is in particular a technique that detects and removes redundancies during the incremental refinement of DDs. This redundancy detection and removal yields substantially smaller DDs.

The DDs studied in the literature and their compilation methods are intrinsically layer oriented. For problems like our prize-collecting job sequencing this layered structure is not natural and prevent to a certain degree compact DDs representing strong relaxations. Therefore, we propose a new compilation approach for relaxed DDs that avoids an explicit consideration of layers. Moreover, we describe a novel construction method for restricted DDs that exploits already gained information from a previously compiled relaxed DD. This work has been submitted to the *INFORMS Journal on Computing*:

M. Horn, J. Maschler, G. R. Raidl, and E. Rönnberg. A*-based construction of decision diagrams for a prize-collecting scheduling problem. *INFORMS Journal on Computing*, submitted.

It is necessary to mention that the development of layer-free compilation of DDs has been primarily conducted by my coauthors. My contributions focused on the construction technique that exploits information from the relaxed DDs. Moreover, as both approaches have been executed in sequence, the author of this thesis was also involved in the testing of the various configurations of both methods.

This chapter is based on the extended version of our initial work and on the advanced approaches proposed in our recently submitted paper.

4.1 Introduction

We consider a new prize-collecting variant of the *job sequencing with one common and multiple secondary resources* (JSOCMSR) problem [55] which we call PC-JSOCMSR. Given a set of jobs, each associated with a prize, the task is to find a subset of jobs with maximal total prize that is feasibly schedulable. Each job requires one of several secondary resources during its whole processing time and a common resource which is shared among all jobs for a part of its execution. Moreover, each job has to be performed within given time windows. Due to these time windows, it is in general not possible to feasibly schedule all jobs.

PC-JSOCMSR originates from the context of particle therapy for cancer treatment (see Chapter 3). In this scenario the common resource corresponds to a particle beam that can be directed into one of multiple treatment rooms which are represented by the secondary resources. Jobs describe treatments that consist of several tasks within a treatment room from which only one is the actual irradiation using the beam. Another application of PC-JSOCMSR is in the pre-runtime scheduling of avionic systems. In this setting we consider a single communication module and multiple applications modules corresponding to the common and secondary resources, respectively. Jobs are performed on one of several applications modules and have to exchange information at their beginning or end using a shared communication module.

In this chapter we explore the potential of applying the concept of *decision diagrams* (DDs) to PC-JSOCMSR. In particular investigate different methods for creating them. Essentially, DDs are rooted directed acyclic multigraphs used to compactly represent a COP's set of feasible solutions. To this end, the nodes of DDs are typically partitioned into layers. The first of the layers contains the root node and each subsequent layer of the DD is associated with one of the decision variables of the COP. Every arc in the DD describes an assignment of the variable represented by the corresponding layer. Thus, a path starting from the root node represents a variable assignment. The lengths of the arcs are assigned in such a way that the length of a path corresponds to the objective value of the corresponding variable assignment. Depending on whether the COP's objective is to maximize or to minimize a given objective function, we are seeking a longest or a shortest inclusion maximal path to a valid terminal node within the DD. The out-degrees of the DD's nodes directly corresponds with the domain sizes of the respective decision variables. If the COP is modeled with binary variables, then all nodes have at most two outgoing arcs and the DD is called *binary decision diagram* (BDD). In the more general case with finite variable domains, the number of arcs leaving nodes is not restricted. In this case, DDs are called *multivalued decision diagrams* (MDDs).

DDs resemble in many aspects a dynamic programming's state graph [54]. Likewise, the size of exact DDs grows in general exponentially with the problem size. To overcome the resulting limitations, Andersen et al. [3] proposed the concept of relaxed DDs. The basic idea is to merge nodes on the same layer and to redirect the affected arcs. This might introduce new paths in the DD that, however, do not represent feasible solutions.

Consequently, relaxed DDs encode a superset of the feasible solutions and represent a discrete relaxation of the problem that provides dual bounds. Another way to cope with the in general exponential number of nodes are restricted DDs [11]. A restricted DD is obtained from an exact DD by removing nodes and all incident arcs. Clearly, this also removes all paths from the DD that included at least one of the removed nodes. Therefore, a restricted DD represents only a subset of all feasible solutions, and it is used to obtain a feasible heuristic solution and a respective primal bound.

The concept of DDs has been successfully applied to a variety of problems, ranging from binary optimization problems to sequencing problems. The former include set covering [9, 11], maximum independent set [10, 13], maximum cut [13], and maximum 2-satisfiability [13] problems and are approached using BDDs. Sequencing problems as our PC-JSOCMSR on the other hand typically suggest the use of MDDs. For this reason, from here on we primarily consider MDDs. Nevertheless, all discussed techniques can be easily adapted to the binary case. Sequencing problems studied in the literature include the time dependent traveling salesman problem with and without time windows and the time-dependent sequential ordering problem [23, 68]. For a comprehensive overview on DDs see [12].

Two main approaches have been proposed for compiling MDDs. The first starts at the root node and constructs the MDD from top down layer by layer [9, 11]. If the number of nodes within a layer exceeds a given limit, then either nodes are merged or removed to obtain a relaxed or a restricted MDD, respectively. The second approach, starts with a simplistic relaxed MDD and applies incremental refinements by splitting nodes in order to iteratively strengthen the relaxation [23, 68]. We start by adapting both approaches for PC-JSOCMSR and are, to our knowledge, the first who directly compare the two techniques experimentally. Our computational experiments show that the relaxed MDDs obtained by the *incremental refinement* (IR) approach provide on most of our benchmark instances better dual bounds than the *top-down construction* (TDC). While the TDC for restricted MDDs outperforms a GVNS metaheuristic on small to medium-sized instances, the GVNS is mostly superior on larger instances.

These two standard approaches to compile relaxed or restricted MDDs are strongly layer oriented, i.e., MDDs are compiled in such a way that nodes can be partitioned into layers and arcs exist only between subsequent layers. However, this strict layer-structure is for problems like the PC-JSOCMSR a major restriction. In PC-JSOCMSR closely related or identical states may frequently be reached from the root node via paths with different numbers of arcs. Such states are in layer-oriented relaxed MDDs represented by multiple nodes located in different layers. This may introduce substantial redundancies and considerably increase the necessary size of a relaxed MDD to achieve a certain quality of the dual bound.

We propose to compile relaxed MDDs in a way that is not restricted to a layered structure and, therefore, offers more flexibility. This is achieved by following the principles of A* search but limiting the number of not expanded nodes. In case the list of unexpanded nodes gets too large, suitable less promising open nodes are merged. To this end, a data

structure of so-called collector nodes is used that allows to efficiently detect promising nodes for merging. Another feature of our MDD construction is that it is guided by a fast-to-calculate, problem-specific upper bound for partial solutions. Compared to the two conventional methods our new A*-based approach allows obtaining smaller relaxed MDDs with stronger bounds in shorter time. Finally, we show how a restricted MDD can be constructed efficiently by exploiting knowledge already contained in the relaxed MDD, in order to return promising heuristic solutions for PC-JSOCMSR instances with up to 500 jobs. Both of the proposed methods contain novel aspects that are also applicable to a broader class of COPs.

The remainder of this chapter is organized as follows. We start in the next section by giving an overview over related literature. Afterwards, we give a formal definition of the considered problem in Section 4.3 and discuss its applications. Section 4.4 provides a recursive model for PC-JSOCMSR which serves as basis for deriving MDDs in Section 4.5. The next five sections are dedicated to the presented compilation methods for either relaxed or restricted MDDs. The first two are adaptions for PC-JSOCMSR of the layer-oriented compilation techniques known from the literature: Section 4.6 describes the TDC of relaxed and restricted MDDs. The IR algorithm for compiling relaxed MDDs is explained in Section 4.7. Afterwards, our A*-based construction method for relaxed MDDs is presented in Section 4.8. To this end, we first describe the novel compilation approach in a problem-independent way and give the aspects specific to PC-JSOCMSR in a second step. In Section 4.9 and Section 4.10 we explain how to further strengthen an obtained relaxed MDD by filtering and how to boost the construction of a restricted MDD by exploiting an existing relaxed MDD. Section 4.11 sketches the standalone GVNS which is used as a reference approach. We further compare our approaches to the MILP and the *constraint programming* (CP) models from Horn et al. [56] which are provided for sake of completeness in Appendix A. Results of the conducted computational experiments are reported in Section 4.12. Finally, Section 4.13 concludes with an outlook on future research directions.

4.2 Related Work

In a complementary work to our initial paper [79], Horn et al. [56] focus on solving the PC-JSOCMSR exactly by means of A* search. They studied different variants of upper bound calculations for partial solutions. These calculations are based on solving different LP relaxations of 0–1 knapsack subproblems and combining results. The A* algorithm was compared to a compact MILP model solved by Gurobi and a MiniZinc CP model solved by three different backends. All approaches were tested on artificially created instances based on real properties from the particle therapy scenario. The A* algorithm outperforms the other approaches clearly and solves instances with up to 40 jobs consistently to optimality. In cases where A* is not able to reach optimality, obtained upper bounds are typically stronger than those from the MILP approach. Nevertheless, the applicability of these methods is strongly limited to rather small or medium sized-problem instances.

More generally, PC-JSOCMSR is an extension of the *job sequencing with one common and multiple secondary resources* (JSOCMSR) problem from Horn et al. [55]. There, no time windows and job prizes are given, all jobs need to be scheduled and the makespan is to be minimized. Horn et al. [55] proposed a strong lower bound calculation for the makespan and utilized this bound in a greedy constructive heuristic and an exact anytime variant of an A* algorithm, which is able to solve instances with up to 1000 jobs to proven optimality.

Van der Veen et al. [110] considered a scheduling problem with a similar scenario as the PC-JSOCMSR. Here jobs also require a common resource as well as an individual secondary resource, but the post-processing times are negligible compared to the total processing times of the jobs. This simplifies the problem substantially since it implies that the start time of each job only depends on its immediate predecessor: If a job j requires a different resource than its predecessor j' then j can always be started so that it takes over using the common resource immediately after job j', and if j requires the same resource as job j' then j can always be started after job j' has finished with its post-processing. Due to these properties, the problem can be modeled as a traveling salesman problem with a special cost structure that allows to solve the problem efficiently in time $O(n \log n)$.

Loosely related to the JSOCMSR are no-wait flowshop problem variants. For a survey, see [2]. In such problems, there are m machines and each job needs to be executed on each of these machines in the same order such that the execution of the job on a successive machine always has to start immediately after the execution ends on the preceding machine. Gilmore and Gomory [40] showed that for two machines this problem can be transformed into a specially structured traveling salesman problem such that the problem can be solved in time $O(n \log n)$. For three or more machines, this problem is NP-hard.

Another possibility is to model the PC-JSOCMSR problem as a more general *resource-constrained project scheduling problem* (RCPSP) (for a survey see Hartmann and Briskorn [49]) with maximal time lags by splitting each job according to the resource usage into three sub-jobs. These three sub-jobs must be executed sequentially without any time lags. However, since we need for each job of the PC-JSOCMSR problem three jobs of the RCPSP and the RCPSP also is known to be difficult to solve in practice, this approach does not seem to be promising to yield good results in practice.

Although the PC-JSOCMSR is an improvement over the simpler JSOCMSR model concerning the practical relevance in the two explained applications, it still is a strongly simplified formulation addressing only certain aspects of the sketched real-world scenarios. Concerning the particle therapy application, Chapter 3 considers more practically relevant aspects including in particular also the planning over many days. Moreover, in contrast to the jobs considered here, DTs can require an arbitrary set of resources. This restriction is motivated by the fact that the beam and the treatment rooms are expected to be the main bottlenecks and correspond here to the common and the secondary resources.

4.3 Problem Description

The *prize-collecting job sequencing with one common and multiple secondary resources* (PC-JSOCMSR) problem is formally defined as follows. Given is a set of n jobs $J = \{1, \ldots, n\}$, a special common resource 0, and a set of m secondary resources $R = \{1, \ldots, m\}$. We denote by $R_0 = \{0\} \cup R$ the complete set of resources. To be processed, each job $j \in J$ requires a resource $q_j \in R$ for its entire processing time $p_j > 0$ and additionally resource 0 for a duration of p_j^0 after time p_j^{pre} from the job's start, with $0 < p_j^0 \leq p_j - p_j^{\text{pre}}$. Hence, the common resource 0 is shared by all jobs whereas each secondary resource is shared by only some jobs. For convenience, we denote with p_j^{post} the duration after the common resource is used until the job j is completed, i.e., $p_j^{\text{post}} = p_j - p_j^{\text{pre}} - p_j^0$.

We associate with each job j a set of time windows $W_j = \bigcup_{w=0,\ldots,\omega_j} [W_{j,w}^{\text{start}}, W_{j,w}^{\text{end}}]$, where $W_{j,w}^{\text{end}} - W_{j,w}^{\text{start}} \geq p_j$. Jobs can only be performed within these time windows and are assumed to be non-preemptive, i.e., may not be interrupted. We denote the whole relevant time horizon, encompassing all time windows of all jobs, with $[T^{\min}, T^{\max}]$.

Finally, each job j has a prize (utility value, priority) $z_j > 0$. We assume that there exists, in general, no feasible schedule that considers all jobs in J. Instead, we aim for a subset of jobs $S \subseteq J$ that can be feasibly scheduled and maximizes the total prize of these jobs, i.e.,

$$Z(S) = \sum_{j \in S} z_j. \tag{4.1}$$

A feasible schedule assigns each job in S a starting time in such a way that each resource is used by at most one job at the same time and that each job is completely performed within one of its time windows in a non-preemptive way. We denote with Z^* the maximum total prize over all feasible solutions, i.e., the optimal solution value.

Figure 4.1 illustrates a possible schedule for a PC-JSCOCMSR instance with $n = 10$ jobs and $m = 3$ secondary resources in which jobs $S = \{1, 4, 7, 8, 10\} \subseteq J$ are scheduled. The horizontal axis shows the time, whereas all resources are represented by the vertical axis. We assume that the remaining jobs cannot be additionally scheduled due to their time windows.

A schedule of S implies a total ordering of the scheduled jobs because all jobs require resource 0 and this resource can be used by only one job at a time. Vice versa, such an ordering $\pi = (\pi_i)_{i=1,\ldots,|S|}$ of S can be decoded by scheduling each job from S in the order given by π at the earliest feasible time after the preceding job. If at least one of the jobs cannot be feasibly scheduled in this way, then ordering π does not represent a feasible solution. We call the schedule obtained from ordering π by the above decoding a *normalized schedule*. Clearly, for every feasible solution there exists a normalized schedule with the same objective value. Hence, we write $Z(\pi)$ for the total prize of the normalized solution given by the ordering π of jobs. The job ordering of the schedule depicted in

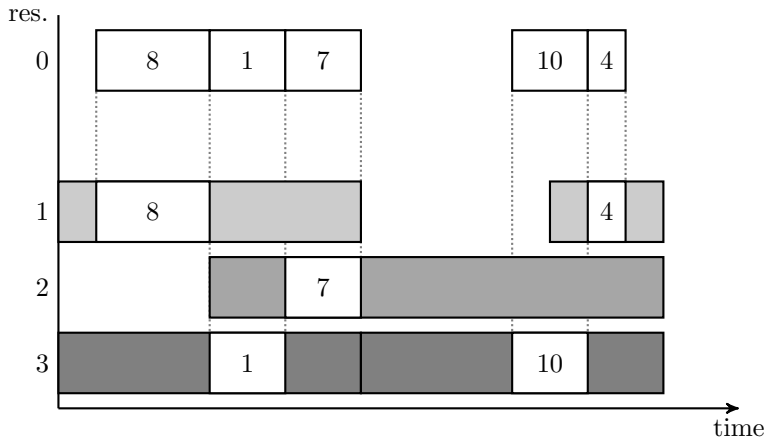


Figure 4.1: Example of a feasible schedule of jobs $S = \{1, 4, 7, 8, 10\}$ of a PC-JSOCMSR instance with $n = 10$ jobs and $m = 3$ secondary resources. The job sequence on the common resource is $(8, 1, 7, 10, 4)$.

Figure 4.1 is $(8, 1, 7, 10, 4)$. Moreover, the schedule shows such a normalized schedule since no single job can start earlier without raising a resource conflict.

As already mentioned PC-JSOCMSR extends the JSOCMSR problem from Horn et al. [55] by time windows and by the aspect that not all the jobs have to be scheduled. In Horn et al.'s JSOCMSR, the objective is to minimize the makespan. They showed that the decision variant of JSOCMSR is NP-hard for $m \geq 2$. PC-JSOCMSR is NP-hard as well, which can be shown by a simple reduction. To this end, we construct an instance for PC-JSOCMSR by associating each job with a single time window $[0, M]$, where M is the given constant for the makespan. There exists a solution for the decision variant of JSOCMSR if and only if there exists a solution for the constructed PC-JSOCMSR instance in which all jobs can be scheduled.

4.3.1 Application in Particle Therapy

One application of the PC-JSOCMSR can be found in the detailed daily scheduling of particle therapies for cancer treatments (see Chapter 3). Here the common resource corresponds to a synchrotron (i.e., a particle accelerator) in which proton or carbon particles get accelerated to almost light speed and are redirected to one of several treatment rooms in which one patient gets irradiated at a time. The treatment rooms are the secondary resources, since each treatment takes place in one room for the whole treatment duration. Each treatment starts with some specific preparations such as positioning and possibly sedating the patient, then the irradiation with the synchrotron—i.e., the usage of the common resource—takes place. Afterwards, some medical examinations need to be done before the patient can leave and the room becomes available for a next patient. Due to the huge costs, there is only a single synchrotron available in a therapy

treatment center and it usually serves two or three treatment rooms[1]. The treatment rooms are usually individually equipped to handle different kinds of treatments, and therefore, patients are pre-assigned to specific rooms. Furthermore, the treatments are constrained by time windows due to the availability of the underlying resources. Ideally, one aims to find a schedule where the synchrotron is directly switched from one treatment room to another so that significant breaks between treatments are avoided. Since only a daily schedule is considered and due to the time windows, it is typically not possible to schedule all treatments. Therefore, a best subset of treatments that can be feasibly scheduled must be selected, and unscheduled patients are treated at other days. The job prizes may correspond to the duration of the irradiations and/or reflect urgencies of the treatments.

4.3.2 Application in Pre-Runtime Scheduling of Avionic Systems

Another context in which the PC-JSOCMSR appears as a sub-structure is in pre-runtime scheduling of electronics within an aircraft, called avionics. In Blikstad et al. [14], the scheduling of an industrially relevant avionic system is addressed. There, the system under consideration consists of a set of nodes and each of these contains a set of modules (processors) with jobs to be scheduled. We address here how partial schedules for the modules in a node can be constructed by solving a PC-JSOCMSR. Compared to Blikstad et al. [14], some simplifications are made with respect to the types of jobs included and by omitting precedence relations between jobs. Also, we do here not explicitly and fully consider the scheduling of the communication network used for communication between the nodes.

In each node, there is a single module called the communication module, which corresponds to the common resource in PC-JSOCMSR. In addition to this module, each node has a set of application modules, which correspond to the secondary resources. Jobs on the application modules are referred to as partition jobs, and they run the system's software applications. We study the case where these jobs use the common resource either at their beginning or at their end. The processing time of a partition job is long compared to that of the other jobs and its use of the common resource is short compared to the total processing time of the job.

There are two types of jobs that use the common resource only, the communication jobs and the regular jobs, and these have short processing times. Both of these types of jobs are involved in handling communication of different kinds (system external, inter- and intranode) but the communication jobs have the specific purpose of representing the jobs used for sending the communication messages, c.f. Blikstad et al. [14]. To represent jobs that use the common resource only, an artificial secondary resource is introduced. Each regular and communication job gets this artificial resource as secondary resource assigned and uses both, the common and the artificial secondary resource, for its whole processing time.

[1]Sometimes four or five treatment rooms are available, but the additional rooms are used as a backup or for special other purposes only.

A characteristic of the communication jobs, and the reason for treating them separately from the regular jobs, is that their time windows originate from time slots where communication messages can be sent. For this reason, there are many time windows for each job and each such time window has a length equal to the processing time of the job. Also, the time windows for the communication jobs can only start at certain points in time, corresponding to the location of the time slots.

To mimic the situation of creating a partial schedule for a node in the system, only a part of the total length of a schedule is considered and the tasks available exceed what is possible to include in the partial schedule. The prize of a job reflects the individual importance of including this job in the partial schedule.

4.4 Recursive Model

We provide a dynamic-programming-like recursive model for PC-JSOCMSR. The induced state graph will then serve as a basis for deriving MDDs. The main components of the recursive model are the states, the control variables that conduct transitions between states, and finally the prizes associated with the transitions. In our recursive formulation a state is a tuple (P, t) consisting of the set $P \subseteq J$ of jobs that are still available for scheduling and a vector $t = (t_r)_{r \in R_0}$ of the earliest times from which on each resource r is available. The initial state corresponding to the original PC-JSOCMSR instance without any jobs scheduled yet is $\mathbf{r} = (J, (T^{\min}, \ldots, T^{\min}))$.

The control variables are $\pi_1, \ldots, \pi_n \in J$. Starting from the root node they select the jobs to be scheduled. Variable π_1 selects the first job j to be scheduled, and we transition from state $\mathbf{r}$ to a successor state (P', t'), where π_2 decides with which next job to continue. This is repeated for all control variables. If a job selected by a control variable cannot be feasibly scheduled as next job, then we obtain the special infeasible state $\hat{0}$. Any further transition from $\hat{0}$ yields $\hat{0}$ again.

The formal specification of the state transitions makes use of the two following definitions. To simplify the handling of time windows let us define the function *earliest feasible time* $\mathrm{eft}(j, t)$ that computes for a given job j and time point t the earliest time not smaller than t at which job j can be performed according to the time windows, i.e.,

$$\mathrm{eft}(j, t) = \min\{T^{\max},\ t' \geq t \mid [t', t' + p_j] \subseteq W_j\}. \tag{4.2}$$

The value $\mathrm{eft}(j, t) = T^{\max}$ indicates that job j cannot be feasibly included in the schedule at time t or later.

Moreover, let the starting time of a next job $j \in J$ w.r.t. a state (P, t) be

$$\mathrm{s}\left((P(u), t(u)), j\right) = \mathrm{eft}\left(j, \max\left(t_0 - p_j^{\mathrm{pre}}, t_{q_j}\right)\right). \tag{4.3}$$

The transition function to obtain the successor (P', t') of state (P, t) when scheduling job $j \in J$ next is

$$\tau((P,t),j) = \begin{cases} (P \setminus \{j\}, t') & \text{if } \mathrm{s}((P,t),j) < T^{\max} \\ \hat{0} & \text{else,} \end{cases} \tag{4.4}$$

with

$$t'_0 = \mathrm{s}((P,t),j) + p_j^{\mathrm{pre}} + p_j^0, \tag{4.5}$$

$$t'_r = \mathrm{s}((P,t),j) + p_j \qquad \text{for } r = q_j, \tag{4.6}$$

$$t'_r = t_r \qquad \text{for } r \in R \setminus \{q_j\}. \tag{4.7}$$

All states except the infeasible state $\hat{0}$ are possible final states. The prize associated with a state transition is job j's prize z_j. Any sequence of state transitions $\tau(\dots\tau(\mathbf{r}, \pi_1)\dots, \pi_i)$ yielding a feasible state (P, t) from the initial state $\mathbf{r}$ represents a solution. In fact, the respective states map directly to the normalized schedule obtained by decoding the jobs $\pi_1, \dots, \pi_i$ as stated in Section 4.3. Moreover, the sum of the prizes of all these transitions corresponds to $Z(\pi_1, \dots, \pi_i)$, the total prize of the solution.

Note that a feasible state does not have to describe a single solution, because the same state might be reached by multiple transition sequences. These different transition sequences yielding the same state might also have distinct total prizes. Since we are maximizing the total prize, we are primarily interested in sequences with maximum total prize. To this end, let $Z^{\mathrm{lp}}(P, t)$ be this maximum total prize for any sequence $\tau(\dots\tau(\mathbf{r}, \pi_1)\dots, \pi_i)$ resulting in state (P, t). Ultimately, we are looking for a feasible state with maximum $Z^{\mathrm{lp}}(P, t)$.

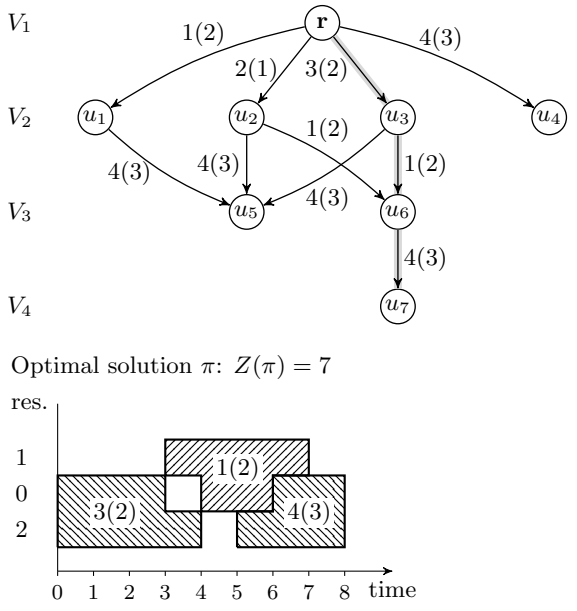
Looking at these relationships from a dynamic programming perspective, we can express the maximum total prize for jobs that can still be scheduled from any feasible state (P, t) onward by

$$Z^*(P,t) = \max\left\{0, z_j + Z^*(\tau((P,t),j)) \mid j \in P,\ \tau((P,t),j) \neq \hat{0}\right\}, \tag{4.8}$$

and $Z^*(\mathbf{r})$ then denotes the overall maximum achievable prize, i.e., the optimal solution value.

Strengthening of States. The individual states obtained by the transitions can be safely strengthened in many cases in order to reduce the number of states to be considered and to avoid infeasible transitions. To this end, let us define the following dominance relation. A state $(P'(u), t'(u))$ *dominates* a state $(P(u), t(u))$, denoted by $(P'(u), t'(u)) \rhd (P(u), t(u))$, when $P'(u) \subseteq P(u)$, $t'_r(u) \geq t_r(u)$ for all $r \in R_0$, and $(P'(u), t'(u)) \neq (P(u), t(u))$.

When constructing the MDD, we can replace a state by a dominating state if it is ensured that the latter still allows for the same feasible solutions. We achieve this



Instance:

j	p_j	p_j^{pre}	p_j^0	q_j	z_j	W_j
1	4	1	2	1	2	$[3, 8]$
2	3	1	1	1	1	$[0, 4]$
3	4	0	3	2	2	$[0, 4]$
4	3	1	2	2	3	$[3, 8]$

States:

V	P	t
r	$\{1, 2, 3, 4\}$	$(0, 0, 0)$
u_1	$\{4\}$	$(6, 8, 5)$
u_2	$\{1, 4\}$	$(4, 3, 3)$
u_3	$\{1, 4\}$	$(4, 3, 4)$
u_4	$\{\}$	$(8, 8, 8)$
u_5	$\{\}$	$(8, 8, 8)$
u_6	$\{4\}$	$(6, 8, 5)$
u_7	$\{\}$	$(8, 8, 8)$

Figure 4.2: A MDD for an example instance with four jobs and two secondary resources.

strengthening as follows. First we consider the earliest starting times $\mathrm{s}((P,t),j)$ for all jobs $j \in P$. Jobs that cannot be feasibly scheduled next can be safely removed from P, i.e., $P' = \{j \in P \mid \mathrm{s}((P,t),j) < T^{\max}\}$. Afterwards, we set the times $t'_r,\ \forall r \in R_0$, to the earliest time resource r is actually used by the jobs in P'. If a resource is not required by any of the remaining jobs then we set the corresponding time t'_r to $T^{\max}$. More formally,

$$t'_0 = \min_{j \in P'} \left(\mathrm{s}((P,t),j) + p_j^{\text{pre}} \right), \tag{4.9}$$

$$t'_r = \begin{cases} \min_{j \in J_r \cap P'} \ \mathrm{s}((P,t),j) & \text{if } J_r \cap P' \neq \emptyset \\ T^{\max} & \text{else} \end{cases} \qquad \forall r \in R, \tag{4.10}$$

where J_r denotes the subset of all jobs in J which require secondary resource $r \in R$. Note that the above state strengthening also ensures that any state for which no feasible extension exists anymore is mapped to the single target state $(\emptyset, (T^{\max}, \ldots, T^{\max}))$.

4.5 Multivalued Decision Diagrams

This section explains the relationships between the state graph of a PC-JSOCMSR problem instance and our exact, relaxed, and restricted MDDs. An exact MDD is a layered directed acyclic multigraph $G = (V, A)$ with node set V and arc set A. The node set V is partitioned into layers $V = V_1 \cup \ldots \cup V_{n+1}$. Each node $u \in V$ is associated with a state $\sigma(u)$ of the recursive formulation from Section 4.4. In particular, the first layer V_1 consists only of a single node associated with the initial state $\mathbf{r}$. Each subsequent

layer V_i contains nodes for all states obtained from feasible state transitions from states associated with nodes in layer V_{i-1}. Moreover, the MDD has arcs for all feasible state transitions in the state graph connecting the corresponding nodes. The length of these arcs are the state transition prizes z_j. The infeasible state $\hat{0}$ and all transitions to it are omitted.

In the literature, a terminal node $\mathbf{t}$ at layer V_{n+1} is defined that corresponds to the end state where no further job can be feasibly scheduled anymore. Since in our case any node represents a valid end state, we deviate here from the literature and do not make explicit use of this target state.

Let us denote by $j(a) \in J$ the job that is considered in the state transition associated with arc $a \in A$. Moreover, let $A^+(u)$ and $A^-(u)$ indicate the set of all incoming and outgoing arcs of a node $u \in V$, respectively. For a node u we write $P(u)$ and $t(u)$ as a shorthand for the set P and vector t of the node's state. In particular, we denote with $t_r(u)$ for a node u the time from which on each resource $r \in R_0$ is available for performing a next job.

An optimal solution is obtained from an exact MDD by determining a longest path from $\mathbf{r}$ to some end node v and scheduling the jobs associated with each arc in the respective order and at the starting times $\mathrm{s}((P,t),j)$. The length of this path, i.e., the sum of the respective arcs' transition prizes, corresponds to the optimal solution value $Z^*(\mathbf{r})$.

Figure 4.2 shows an exact MDD for an instance with four jobs and two secondary resources. Details of the PC-JSOCMSR instance are given on the top right, while the MDD is depicted on the top left. Each arc's label indicates the job that is scheduled by the respective state transition and in parentheses the arc's length. We indicate with gray-highlighted arcs the in our case unique longest path of length seven. The corresponding optimal solution, scheduling the jobs $\pi = (3, 1, 4)$ with a total prize of $Z(\pi) = 7$, is shown on the bottom left. Moreover, states of all nodes are given on the bottom right.

Observe that arcs exist only between directly successive layers and there might be nodes for identical states on different layers. Later in Section 4.8 we consider the compilation of MDDs that are not explicitly partitioned into layer. This allows avoiding these redundant states on different layers.

Nevertheless, exact MDDs grow in general exponentially with the problem size as they basically represent the complete state graph. We are more interested in more compact MDDs that represent the state graph only in an approximate way. This is usually done by limiting the number of nodes allowed in each layer to a fixed maximum $\beta \geq 1$. The number of nodes in a layer is called the layer's width, and the maximum width over all layers is the width of an MDD. To receive MDDs of limited width, there have been proposed two concepts with contrary effects: *relaxed MDDs* [3] and *restricted MDDs* [11].

Relaxed MDDs cover all feasible solutions as a subset plus possibly a set of solutions that are invalid for the original problem. Thus, they represent a discrete relaxation of

(a) Relaxed MDD

(b) Restricted MDD

Figure 4.3: A relaxed and a restricted MDD for the example instance in Fig. 4.2.

the original problem, and the length of a longest path of a relaxed MDD is a dual bound to the original problem's optimal solution value $Z^*(\mathbf{r})$. To have limited width, a relaxed MDD in general superimposes states of the original state graph: Sets of nodes of an exact MDD are combined into so-called *merged nodes*; all affected arcs are redirected to the respective merged node. To ensure that a valid relaxation is obtained, the state of a merged node must be set so that it is in no dimension stricter than each original state. In case of our PC-JSOCMSR, if a set M of original states is merged, the state of the respective merged node is

$$\oplus(M) = \left(\bigcup_{(P,t)\in M} P, \left(\min_{(P,t)\in M} t_r \right)_{r\in R_0} \right). \tag{4.11}$$

If possible, the obtained state is then strengthened as described in Section 4.4. The merged state allows all feasible extensions that the original states did. It is therefore valid in the sense that no feasible solutions are lost. Additional extensions and originally infeasible solutions may, however, become feasible due to the merge, as is usually the case for relaxed MDDs.

Figure 4.3(a) shows for the exact MDD in Figure 4.2 a relaxed MDD where nodes u_3 and u_4 are merged resulting in node u'. The width of the relaxed MDD decreases from four to three. Recall that the optimal solution of the considered instance has a total prize of seven. The longest path within the relaxed MDD, indicated by the gray-highlighted arcs, has a total length of eight. This is achieved by scheduling job 4 twice, which clearly does not correspond to a feasible solution of the original problem. Moreover, notice that the relaxed MDD contains all paths from the exact MDD. The original optimal solution is still represented by a respective path, however, it is not a longest anymore. The state of the merged node is given by $(\{1,4\},(4,3,4))$, while the states of all remaining nodes do not change.

Restricted MDDs are the second option for approximate MDDs with limited width. They are obtained by removing nodes from an exact MDD with all incoming and outgoing arcs. Whenever a node is removed, also all paths containing the node are not anymore encoded in the MDD. Consequently, a restricted MDD represents only a subset of all feasible solutions, and the length of a longest path in a restricted MDD might be shorter than one in an exact MDD. For this reason the length of a longest path in a restricted MDD is a primal bound to the original problem's optimal solution value $Z^*(\mathbf{r})$.

A restricted MDD for the exact MDD from Figure 4.2 is depicted in Figure 4.3(b). The node u_3 and all its incoming and outgoing arcs are removed. All other nodes, arcs, and states remain unchanged. The longest path in the restricted MDD, again indicated by arcs highlighted in gray, has a total length of six. This longest path encodes a feasible solution to the original problem, however, not an optimal one.

4.6 Top-Down Construction of Decision Diagrams

The *top-down construction* (TDC) [13, 11, 10] compiles exact MDDs, as well as relaxed and restricted MDDs by traversing the state graph in a breadth-first fashion. The method starts with an empty first layer V_1 and adds a node for the initial state $\mathbf{r}$. Then, one layer after the other is filled with nodes. For a subsequent layer V_i, this is done by adding nodes for all feasible states that can be obtained by a transition from any node $u \in V_{i-1}$, i.e., we add a node to V_i for each state in

$$\{\tau(\sigma(u), j) \mid u \in V_{i-1}, j \in P(u)\}. \tag{4.12}$$

Note that identical states produced by different transitions are represented by a single common node within a layer. In addition to the nodes, we also add corresponding arcs for each of the conducted transitions.

When we are compiling relaxed or restricted MDDs, we have to check at this point the width of the current layer V_i. If it exceeds a given maximum β, nodes have either to be merged or dropped, respectively. The quality of the obtained primal and dual bounds from the produced relaxed and restricted MDDs is predominantly influenced by the strategy to select the nodes for merging or removal. The basic idea is to prefer nodes for merging or removal that are unlikely part of any optimal solution. Bergman et al. [10] considered three different merging heuristics: random nodes, nodes with the shortest longest path $Z^{\text{lp}}(u)$, and nodes with the most elements in $P(u)$. In their experiments the second strategy achieved the best results. Moreover, [11] suggest the same node selection heuristic for the compilation of restricted MDDs. We observed that merging or removing nodes with the smallest $Z^{\text{lp}}(u)$ values is disadvantageous for PC-JSOCMSR. This can be explained by the fact that this strategy focuses just on the longest path, but does not respect how well the jobs fit next to each other. Therefore, we set the longest path to a node into perspective with the time the common resource is occupied by the corresponding jobs. The nodes within the currently considered layer V_i, $i > 0$, are sorted according to the ratio $Z^{\text{lp}}(u)/t_0(u)$ in increasing order. We then merge respectively

remove the first nodes until the width of V_i becomes β. Afterwards, we continue with the next layer. The algorithm terminates when either no further state transitions are possible or we completed layer V_{n+1}.

4.7 Incremental Refinement of Relaxed Decision Diagrams

The basic idea of an IR approach is to apply filtering and refinement steps iteratively on an initial simple relaxed MDD in order to improve it and approximate an exact MDD. Filtering steps remove arcs that are only contained in root to sink paths that represent infeasible solutions. The refinement steps consist of splitting nodes to represent so far merged states in more detail and as a consequence to trigger further filtering of arcs. The main goal of IR is to decrease the length of longest paths in the MDD, i.e., the obtained upper bound on an instance's solution value.

IR has been initially proposed by Hadzic et al. [45] and Hoda et al. [51] for constraint satisfaction systems. The central aspect of this approach is the division of filtering and refinement into independent operations. As a consequence, the overall algorithm can apply and combine these operations however it is appropriate. A relaxed MDD for the PC-JSOCMSR problem contains in general paths that do not represent feasible solutions, either because jobs occur more than once or not all jobs can be scheduled within their time windows. Therefore, we have to find refinement and filtering operations that allow us to exclude job repetitions and time window violations.

Due to the fact that exact MDDs have in general an exponential number of nodes w.r.t. the problem size, we cannot hope to apply refinement and filtering until all invalid paths are sorted out for problem instances of practically relevant size. Hence, a key aspect of an IR approach is the order in which the refinement steps are applied on the nodes. The works from [23] and [68] provide an IR method for sequencing problems in which a permutation of jobs has to be found. Essentially, they order the jobs according to the processing times and with it to a certain extent according to the length of the corresponding arcs within the MDD. Their approach removes repetitions of jobs according to that order until the maximal allowed width of the MDD is reached. The rationale behind this strategy is that repetitions of jobs represented by long arcs are more frequently contained within longest paths. For PC-JSOCMSR this method is, however, not suitable because we have to assume that only a fraction of the jobs can be actually scheduled. Hence, it is not clear in advance which jobs play a key role for deriving a good approximation of an exact MDD.

Our IR for PC-JSOCMSR uses a current longest path as guidance. We follow the arcs on such a longest path, starting from the root node, and check for each arc whether the associated job can be feasibly scheduled. In case that a job occurs more than once, we refine the MDD s.t. repetitions of this job are not possible anymore. If a job cannot be feasibly scheduled within its time windows, we split nodes to allow excluding this path.

Algorithm 4.1: *Incremental refinement guided by longest paths* (IRLP)

```
   Input: initial relaxed MDD G = (V, A) with V = V_1 ∪ ··· ∪ V_{n+1}
 1 while termination criterion not met do
 2     Let p be a longest path in G;
 3     if p admits a feasible schedule then
 4         return;                                  // optimal solution has been found
 5     if p contains a job repetition then
 6         for i ← 2 to n + 1 do
 7             foreach node u ∈ V_i do
 8                 update node u and filter incoming and outgoing arcs;
 9                 split node u into two if it allows to avoid the node repetition;
10             merge nodes with identical states in V_i;
11     else                                         // p contains a time window violation
12         Split nodes on p to avoid the identified time window violation;
13         for i ← 2 to n + 1 do
14             foreach node u ∈ V_i reachable from the split nodes do
15                 update node u and filter incoming and outgoing arcs;
16             merge nodes with identical states in V_i;
```

Algorithm 4.1 shows an outline of the proposed *incremental refinement guided by longest paths* (IRLP). It acts on a given relaxed MDD, which is obtained in our case by the top down construction from Section 4.6 with a small initial width. In each iteration of the main while loop we obtain a longest path. If the sequence of jobs represented by the path can be feasibly scheduled, then we have found an optimal solution and terminate.

Otherwise, depending on whether we detected a job repetition or a time window violation on the currently considered longest path the following steps differ. In the former case we traverse the MDD starting from the root node **r** layer by layer. For each considered node we try to filter arcs and update the node's state if necessary. We check next if the node has to be refined and perform a node split if it allows to remove the considered job repetition. After all nodes of a layer have been considered, we might encounter that nodes are associated with an identical state. Such nodes are merged to avoid redundancies in the MDD. In the latter case of a time window violation we perform a much more local refinement operation in which only nodes along the considered path are split. In the subsequent filtering we consider all nodes reachable from the previously split nodes. We enforce also here that all nodes within each layer represent a distinct state. Notice that the refinement of job repetitions is preferred over the refinement of time window violations if both are contained in the longest path. This has shown to provide better bounds if a binding time limit is given as termination criterion. The applied filtering techniques and the updating of the nodes' state are described in Section 4.7.1. The

two types of refinement operations are presented in more detail in Section 4.7.2 and Section 4.7.3. Finally, Section 4.7.4 explains how we avoid producing nodes representing identical states within layers.

4.7.1 Node Updates and Filtering

Filtering applied in an IR method aims at identifying and removing arcs that are only contained in paths corresponding to infeasible solutions. The filtering techniques generally rely on the Markovian property of the MDD's states, which means that a state is defined by its predecessors and the transitions. This allows specifying tests that use information local to a considered node to decide whether incoming or outgoing arcs can be removed.

An intrinsic part of the presented filtering method is to keep the node's states always up to date, which is necessary because the removal of a node's incoming arcs may change its associated state. Moreover, an adjustment of a node's state may imply further changes on the nodes reachable from the currently considered node. Therefore, we traverse the MDD s.t. we reach a node after we have processed all its predecessors. Consequently, we end up in each iteration of the IRLP with an MDD where all states fulfill the Markovian property. For each considered node we first update the node's state and then check whether incoming or outgoing arcs can be removed. In case incoming arcs are removed the node's state has to be reevaluated again. An update of a state consists of reassessing and merging the transitions from all predecessors, which means for a node u to compute

$$\oplus \left(\{ \tau(\sigma(v), j(a)) \mid a = (v, u) \in A^+(u) \} \right). \tag{4.13}$$

Such a state update is a computational expensive operation and should only be performed if a node's state may actually change. For this reason, we recompute a node's state only if either a predecessors state has changed or if an incoming arc has been removed.

Let (P, t) and (P', t') be node u's state before and after a reevaluation, respectively. Due to the definition of the relaxation scheme (4.11) and the fact that we are only removing arcs during filtering, it holds that $t'_r \geq t_r$ for all $r \in R_0$ and $P' \subseteq P$. In case $P' \subset P$, we remove all outgoing arcs $a \in A^-(u)$ with $j(a) \notin P'$ since they cannot be part of any feasible solution represented by a path reaching u from $\mathbf{r}$. If any node except $\mathbf{r}$ ends up without any incoming arc, it is removed together with all its outgoing arcs.

4.7.2 Refinement of Job Repetitions

We discuss in this section a technique that modifies an MDD in such a way that a considered job j occurs on each path at most once. This method is conceptually an adaptation from the one proposed by Cire and Hoeve [23], but takes into account that in PC-JSOCMSR usually only a subset of the jobs can be scheduled. The refinement is based on the observation that a job repetition occurs if a job j is contained on a path starting from node $\mathbf{r}$ to a node u and job j is still included in $P(u)$. Consequently, node u has an outgoing arc associated with job j which represents a repetition. Before we

can derive a splitting strategy, we first have to verify if the above condition is sufficient to detect all job repetitions. To this end we denote with $\text{Some}_u^{\downarrow} \subseteq J$ the subset of jobs appearing in some path from $\mathbf{r}$ to a node $u \in V$. For a node $u \in V$ the set $\text{Some}_u^{\downarrow}$ can be calculated recursively by

$$\text{Some}_u^{\downarrow} = \bigcup_{a=(v,u)\in A^+(u)} \left(\text{Some}_v^{\downarrow} \cup \{j(a)\}\right). \tag{4.14}$$

We show next that we can determine repetitions of a considered job j occuring on some path in a MDD by using $P(u)$ and $\text{Some}_u^{\downarrow}$ of the nodes u in the MDD.

Lemma 1. *A job j is assigned on each path starting from $\mathbf{r}$ at most once if and only if $j \notin \text{Some}_u^{\downarrow} \cap P(u)$ holds for all nodes $u \in V$.*

Proof. Assume first that a job j is associated with at most one arc in every path starting from $\mathbf{r}$ of a given MDD G and consider an arbitrary node $u \in V$. If no path from $\mathbf{r}$ to u has an arc labeled j then it holds by definition that $j \notin \text{Some}_u^{\downarrow}$ and consequently $j \notin \text{Some}_u^{\downarrow} \cap P(u)$. If on the other hand there exists a path from $\mathbf{r}$ to u with an arc associated with j then no path starting from u can contain an arc labeled j. Moreover, it holds by definition that a node $v \in V$ can only have an outgoing arc a with $j(a) = j$ if $j \in P(u)$. Therefore, $j \notin P(u)$ and $j \notin \text{Some}_u^{\downarrow} \cap P(u)$.

Conversely, suppose that $j \notin \text{Some}_u^{\downarrow} \cap P(u)$ for all nodes $u \in V$. In case $j \notin \text{Some}_u^{\downarrow}$ we cannot have a repetition of node j on any path from $\mathbf{r}$ to u. If a node u is reached by an arc associated with job j then $j \in \text{Some}_u^{\downarrow}$ and thus, $j \notin P(u)$. Since node u can have only outgoing arcs for the jobs in $P(u)$, node u cannot have an outgoing arc labeled j. Moreover, since $j \in \text{Some}_v^{\downarrow}$ for all nodes v reachable from node u we can conclude by the same argument that also for these nodes $j \notin P(u)$ and hence there are no respective outgoing arcs. Thus, job j is assigned on each path starting from $\mathbf{r}$ at most once. □

Whenever we detect a node repetition, i.e., $j \in \text{Some}_u^{\downarrow} \cap P(u)$ for some node u, we perform a node split to obtain a node u_1 with $j \notin P(u)$ and a node u_2 with $j \notin \text{Some}_u^{\downarrow}$ as follows.

Theorem 2. *Given job j and a MDD, we replace all nodes $u \in V$ with $j \in \text{Some}_u^{\downarrow} \cap P(u)$ by two nodes u_1 and u_2, s.t. all incoming arcs $a = (v, u)$ are redirected to u_1 if $j \notin P(\tau(\sigma(v), j(a)))$ and to u_2 otherwise. All outgoing arcs are replicated for both nodes. The resulting MDD satisfies $j \notin \text{Some}_u^{\downarrow} \cap P(u)$ for all nodes $u \in V$.*

Proof. For the root node $\mathbf{r}$ we have by definition that $\text{Some}_{\mathbf{r}}^{\downarrow} = \emptyset$ and, thus, $j \notin \text{Some}_u^{\downarrow} \cap P(u)$. Assume as induction hypothesis that the desired condition $j \notin \text{Some}_u^{\downarrow} \cap P(u)$ holds for all predecessors of a node u. In addition, consider that we have replaced node u by the nodes u_1 and u_2 as described above. From the relaxation scheme (4.11) we know that set P of node u_1 cannot contain j. For all of u_2's incoming arcs $a = (v, u_2)$ it holds that $j \notin \text{Some}_v^{\downarrow}$ since otherwise $P(\tau(\sigma(v), j(a)))$ could not contain j. Consequently, u_1 as well as u_2 satisfy the stated condition. □

The actual refinement is done by enforcing Lemma 1 in a single top down pass. To this end, we start with the root node and process all nodes layer by layer. For each considered node u we first update its state if needed and apply the filtering as described in Section 4.7.1. Afterwards, we determine the set $\text{Some}_u^{\downarrow}$ and split node u as described in Theorem 2 if necessary. Whenever a node is split, new states are calculated for the two new nodes. Furthermore, we perform filtering on the new nodes' incoming and outgoing arcs.

4.7.3 Refinement of Time Window Violations

Let sequence $(u_1, a_1, u_2, \ldots, u_k, a_k, u_{k+1})$ of alternating nodes and arcs denote a path in our MDD starting at the root node $\mathbf{r}$ (i.e., $u_1 = \mathbf{r}$) where $(u_1, a_1, u_2, \ldots, u_{k-1}, a_{k-1}, u_k)$ corresponds to a feasible solution but the job represented by arc a_k cannot be additionally scheduled within its time windows. For the considered path we denote with $(u_1^{\downarrow}, \ldots, u_k^{\downarrow})$ the not relaxed states along the considered path. That is, $u_1^{\downarrow} = \mathbf{r}$ and $u_i^{\downarrow} = \tau(u_{i-1}^{\downarrow}, j(a_{i-1}))$ for $1 < i \leq k$. Due to the state relaxations of the nodes in the MDD we observe that $j(a_k) \in P(u_k)$ but $j(a_k) \notin P(u_k^{\downarrow})$. The basic idea is to split the nodes on the path in such way that job $j^{(k)}$ can be removed from $P(u_k)$ and with it also the arc a_k.

In general, it is not sufficient to just split node u_k but a subset of the path's nodes $u_l, \ldots, u_k$, with $1 < l \leq k$, has to be refined. Ideally, the number of nodes to be refined should be small and the refinement should exclude other time window violations as well. We compute the subset of nodes to be refined as follows: We first check whether $s(\tau(\sigma(u_{k-1}), j(a_{k-1})), j(a_k))$ evaluates to $T^{\max}$. If it does, then job $j(a_k)$ cannot be feasibly scheduled on the state resulting from the transition from state u_{k-1}. Consequently, it suffices to refine node u_k. If it does not, then we consider one predecessor more, i.e., we check whether $s(\tau(\tau(\sigma(u_{k-2}), j(a_{k-2})), j(a_{k-1})), j(a_k))$ results in $T^{\max}$. This step is repeated until we find a node u_{l-1} on the considered path which allows excluding job $j(a_k)$ if we follow exact transitions from it.

The actual refinement works as follows: We replace each node u_i with $i = l, \ldots, k$ by nodes $u_{i,1}$ and $u_{i,2}$. The incoming arcs $a = (v, u_i) \in A^+(u_i)$ are redirected to $u_{i,1}$ if $t_r(\tau(\sigma(v), j(a))) \geq t_r(\tau(\sigma(u_{i-1}), j(a_{i-1})))$ for all $r \in R_0$, otherwise, they are redirected to $u_{i,2}$. Outgoing arcs of u_i are replicated for $u_{i,1}$ and $u_{i,2}$. After a node split we determine for the two resulting nodes the corresponding states and perform a filtering of their incoming and outgoing arcs as described in Section 4.7.1. Last but not least, we have to possibly reevaluate the states and filter all incident arcs of all nodes reachable from each node u_i.

4.7.4 Duplicate State Elimination

A side effect of the node updates and the refinement methods is that we might end up with multiple nodes within one layer that represent an identical state. For example, assume that a considered layer already contains a node u without any outgoing arcs that represents state $(\emptyset, (T^{\max}, \ldots, T^{\max}))$. Moreover, suppose that due to updating another

node u' we encounter that its state cannot be feasibly extended and remove all outgoing arcs. Both, u and u', then represent the same identical state, but are reached by different paths. We can avoid this redundancy in the MDD by redirecting all incoming arcs from u' to u and removing u' from V.

In the more general case, where u and u' have outgoing arcs merging nodes with duplicate states is more involved. First of all, we have to ensure that our MDD still remains a valid relaxation. The Markovian property implies that for all feasible extensions of u there exist an equivalent extension of u' and vice versa [23]. This allows us to remove node u' including its outgoing arcs after redirecting all incoming arcs to u. Obviously, the state of node u remains valid. However, if not done carefully, this operation may reintroduce paths encoding infeasible solutions which have been already excluded. Furthermore, we have to make sure that the duplicate state elimination does not produce cycles with the refinement operations in order to guarantee that IRLP terminates. After performing the refinements of job repetitions for a job j on a considered layer it holds that if a job $j \in P(u)$ then $j \notin \text{Some}^{\downarrow}_{u'}$ whenever the states of u and u' are identical. Splitting nodes for refining time window violations aims at increasing the t_r values to trigger filtering. Since our duplicate state elimination does not change the nodes' states, we will from a theoretical point of view always converge to an exact MDD.

Our duplicate state elimination works as follows: After we have performed all refinement, updating, and filtering operations within a layer, we consider all nodes with duplicate states pairwise and remove one of them until all nodes' states are distinct. To this end, we redirect all incoming arcs to the node u having the larger $Z^{\text{lp}}(u)$ value and remove the other node including outgoing arcs. The intention for selecting the node with the larger $Z^{\text{lp}}(u)$ value is that we do not increase the longest path to any node. Moreover, the nodes reachable from u are more likely to be already refined, as IRLP focuses on longest paths.

4.8 A*-Based Construction of Relaxed Decision Diagrams

This section is dedicated to a novel construction scheme for relaxed DDs inspired by A* search where a list of nodes that still need to be processed is maintained. In each iteration a node is selected according to a priority function, removed from the list, and expanded. Merging is performed only with nodes considered similar w.r.t. a labeling function. The key characteristics of this scheme are that it allows merging across layers and that it exploits a problem-specific upper bound function. The aim is to obtain compact relaxed DDs that provide tight upper bounds and that are promising for the further application of primal heuristics, DD-based branch-and-bound, or advanced constraint propagation in CP. We motivate the design of the construction scheme as follows.

The concepts of the TDC and the IR for compiling relaxed DDs have the implicit assumption that node merges are only meaningful on the same layer. However, especially for problems with subset selection scenarios where the cardinality of complete/final solutions is not fixed, merging nodes only within the same layer represents a severe

limitation. In such problems strongly related or even identical states may be reached after different numbers of transitions from the root node, while states reached with the same number of transitions may be more different. PC-JSOCMSR is an example for these properties: Feasible solutions may consider different numbers of jobs and the similarity of states is more related to the times from which on further jobs can be scheduled than the number of so far scheduled jobs. Moreover, in PC-JSOCMSR's setting it is possible that multiple nodes belonging to different layers have the same possibilities for further extensions, and thus, the same sets of paths leading from these nodes further to the target node. This implies that the strict use of layers causes a redundancy in the DD, and that it would be better to represent the respective nodes just by a single node. The possibility of node merges across different layers may open a new dimension of flexibility, possibly allowing to derive much more compact relaxed DDs yielding stronger bounds. Therefore, our approach avoids the explicit consideration of layers.

Another motivation to develop a new construction principle is that for many problems there already exist—or it may not be difficult to come up with—some fast-to-calculate problem-specific upper (dual) bound calculations for partial solutions. It appears natural and promising to also make use of such a function to construct a possibly stronger relaxed DD. In case of PC-JSOCMSR Horn et al. [56] investigated different fast-to-compute upper bounds on the still achievable total prize for scheduling jobs from $P(u)$, given state $(P(u), t(u))$. We adopt here the strongest of these upper bound functions, denoted by $Z^{\mathrm{ub}}(u)$, which is based on solving a set of linear programming relaxations of knapsack problems. For the sake of completeness Appendix A.1 repeats details on how this bound is calculated.

The concept of an A*-based compilation method for relaxed DDs is highly promising also for other problems. Therefore, we present our approach in the next subsection in a problem-independent way and provide the problem-related details afterwards. Although we finally construct MDDs, the approach is also applicable for the compilation of BDDs. To highlight this broader applicability we intentionally refer in the problem-independent part to DDs and omit the distinction between MDDs and BDDs. As PC-JSOCMSR is a maximization problem, we assume maximization. However, a generalization to minimization problems is straightforward. Later, in Section 4.8.2 we discuss the problem-specific details for applying the novel construction scheme to PC-JSOCMSR.

4.8.1 General Scheme

The core construction idea is to apply the principles of classical A* search [48] but to limit the set of open nodes (i.e., not yet expanded nodes) by occasionally merging nodes, selected according to some criterion. We maintain a so-called *open list* Q that is a priority queue containing all open nodes. Each node in the open list $u \in Q$ has assigned a priority

$$f(u) = Z^{\mathrm{lp}}(u) + Z^{\mathrm{ub}}(u), \tag{4.15}$$

where $Z^{\mathrm{lp}}(u)$ denotes the length of a longest so far known path from the root node $\mathbf{r}$ to node u, while $Z^{\mathrm{ub}}(u)$ refers to a problem-specific upper bound for the length of any

feasible path from u to the target node $\mathbf{t}$. Classical A* search always selects a node with the highest priority from the open list Q and expands it until the target node $\mathbf{t}$ gets selected. If $Z^{\text{ub}}(u)$ is indeed a valid upper bound function (which is also called *admissible heuristic*), A* is guaranteed to have found a longest path from the root node $\mathbf{r}$ to the target node $\mathbf{t}$, which corresponds to an optimal solution of the underlying problem.

If we continue the execution of the A* algorithm until the open list Q is empty and store all nodes and arcs for feasible transitions, the algorithm yields a complete exact DD. To obtain a relaxed DD, we restrict the size of the open list to a maximum allowed number of nodes ϕ and merge less promising nodes from Q if necessary. Clearly, this merging of nodes will, in general, prevent the method from reaching proven optimality, but the size of the obtained relaxed DD and the time for compilation will be smaller.

While a complete relaxed DD is obtained only when the algorithm is performed until the open list gets empty, the best upper bound, denoted by $Z^{\text{ub}}_{\min}$, will be obtained when the target node is selected for expansion for the first time. The validity of the upper bound $Z^{\text{ub}}_{\min}$ follows directly from the A* search's optimality property. When continuing the expansion of open nodes, the length of the longest path in the relaxed DD may only become larger due to further merges. Thus, if one only aims at using the DD compilation to obtain an upper bound, the approach can be terminated at this point.

Since the open list Q in general contains nodes of different layers, the A*-based construction algorithm is able to merge nodes across different layers. In particular, if a state can be reached via multiple paths with different numbers of arcs, this is naturally recognized and the state is only represented by a single node. In contrast, a classical TDC or IR would represent the same state in each layer where it occurs by individual nodes. Moreover, in case that the algorithm only expands nodes that have not been obtained through merging until the target node $\mathbf{t}$ is selected, then the longest path corresponds to the classical A* result, and a feasible and proven optimal solution is obtained. Thus, for easier or simpler problems, our approach can behave similarly efficient and optimal as a conventional A* search.

Algorithm 4.2 shows the proposed A*-based construction of a relaxed DD in pseudo-code. First, the open list Q is initialized with the root node $\mathbf{r}$, and $Z^{\text{ub}}_{\min}$ is set to the value obtained from the problem-specific upper bound function Z^{ub} applied to the root node. In each major iteration, a node u with maximum priority $f(u)$ is taken from the open list Q. This node's f value provides a valid upper bound until the target node $\mathbf{t}$ is chosen for expansion. Consequently, $Z^{\text{ub}}_{\min}$ is updated in Line 9 when an improved bound is found. If node u is the target node $\mathbf{t}$, the algorithm may optionally terminate early, returning an incomplete relaxed DD and the upper bound $Z^{\text{ub}}_{\min}$ at Line 12.

The next step differs if node u is already expanded (i.e., has outgoing arcs) or not. If not, the expansion is performed by considering each feasible transition from state $\sigma(u)$ yielding a respective successor state Σ. For each such state Σ a node v is created, if no corresponding node with $\sigma(v) = \Sigma$ exists in V yet. Then, a new arc (u, v) is inserted into A to represent this transition. In case that the new path to v via this arc (u, v) increases

Algorithm 4.2: A*-based construction of a relaxed DD

Input: open list size limit ϕ

Output: relaxed DD $G = (V, A)$ and upper bound to the optimal solution value

$\mathbf{r} \leftarrow$ node corresponding to initial state;
$Q \leftarrow \{(\mathbf{r}, f(\mathbf{r}) = Z^{\text{ub}}(\mathbf{r}))\}$;
$Z^{\text{ub}}_{\text{min}} \leftarrow Z^{\text{ub}}(\mathbf{r})$;
$\mathbf{t}$-expanded $\leftarrow$ *false*;
while $Q \neq \emptyset$ **do**
  $u \leftarrow \arg\max_{u \in Q} f(u)$;
  $Q \leftarrow Q \setminus \{u\}$;
  if not $\mathbf{t}$-expanded **then**
    $Z^{\text{ub}}_{\text{min}} \leftarrow \min(Z^{\text{ub}}_{\text{min}}, f(u))$;
  if $u = \mathbf{t}$ **then**
    $\mathbf{t}$-expanded $\leftarrow$ *true*;
    return incomplete DD G and $Z^{\text{ub}}_{\text{min}}$; *// optional, if only $Z^{\text{ub}}_{\text{min}}$ is needed*
  if u not yet expanded **then** *// expand node u*
    foreach feasible successor state Σ of $\sigma(u)$ **do**
      if $\nexists v \in V \mid \sigma(v) = \Sigma$ **then**
        add new node v to V with $\sigma(v) = \Sigma$, $Z^{\text{lp}}(v) = 0$;
      add new arc $a = (u, v)$ to A;
      if $Z^{\text{lp}}(u) + \text{z}(a) > Z^{\text{lp}}(v)$ **then**
        $Z^{\text{lp}}(v) \leftarrow Z^{\text{lp}}(u) + \text{z}(a)$;
        $Q \leftarrow Q \cup \{(v, f(v) = Z^{\text{lp}}(v) + Z^{\text{ub}}(v))\}$;
  else *// re-expand node u*
    foreach arc $a = (u, v) \in A$ **do**
      if $Z^{\text{lp}}(u) + \text{z}(a) > Z^{\text{lp}}(v)$ **then**
        $Z^{\text{lp}}(v) \leftarrow Z^{\text{lp}}(u) + \text{z}(a)$;
        $Q \leftarrow Q \cup \{(v, f(v) = Z^{\text{lp}}(v) + Z^{\text{ub}}(v))\}$;
  if $|Q| > \phi$ **then** *// reduce size of Q*
    try to merge nodes in Q until $|Q| \leq \phi$ according to Algorithm 4.3;
return relaxed DD G and upper bound $Z^{\text{ub}}_{\text{min}}$;

$Z^{\text{lp}}(v)$, then node v is inserted into Q. If node u was already expanded, a re-expansion has to take place because a longer path to u, yielding a larger $Z^{\text{lp}}(u)$, has been found in an iteration after the node's original expansion. Note that in general we cannot avoid such re-expansions even when the upper bound function is consistent since node merges may lead to new longer paths. The re-expansion is done by propagating the updated $Z^{\text{lp}}(v)$ to all its successor nodes and if their Z^{lp} value increases as well then they are added to the open list for re-expansion. After each node expansion, the algorithm checks if the size of the open list $|Q|$ exceeds the limit ϕ. If this is the case, then the algorithm tries to reduce Q by merging nodes as shown in Algorithm 4.3 and explained in the next paragraphs. Algorithm 4.2 terminates when the open list gets empty by returning the relaxed DD together with the best obtained upper bound $Z^{\text{ub}}_{\text{min}}$. Note that $Z^{\text{ub}}_{\text{min}}$ may be smaller than the length of a longest path in the final DD (i.e., $Z^{\text{lp}}(\mathbf{t})$), due to node merges after selecting the target node for expansion.

Algorithm 4.3: Reduce size of open list

Input: open list Q, global set of collector nodes V^{c} (initially empty)
Output: possibly reduced open list Q

for $u \in Q$ in increasing order of $Z^{\text{lp}}(\cdot)$ values **do**
  if $|Q| \leq \phi$ **then**
    break;
  while u not expanded $\wedge\ \exists v \in V^{\text{c}} \mid L(v) = L(u) \wedge u \neq v \wedge$ v not expanded **do**
    create node v' with merged state $\sigma(v') = \oplus(\{\sigma(u), \sigma(v)\})$;
    $Q \leftarrow Q \setminus \{u, v\}$;
    $V^{\text{c}} \leftarrow V^{\text{c}} \setminus \{v\}$;
    if $\exists v'' \in V \mid \sigma(v'') = \sigma(v')$ **then**
      $f''_{\text{old}} \leftarrow f(v'')$;
      redirect all incoming arcs from v' to v'';
      if $f(v'') > f''_{\text{old}}$ **then**
        $Q \leftarrow Q \cup \{(v'', f(v'') = Z^{\text{lp}}(v'') + Z^{\text{ub}}(v''))\}$;
      $u \leftarrow v''$;
    else
      $V \leftarrow V \cup \{v'\}$;
      $Q \leftarrow Q \cup \{(v', f(v') = Z^{\text{lp}}(v') + Z^{\text{ub}}(v'))\}$;
      $u \leftarrow v'$;
  if u not expanded **then**
    $V^{\text{c}} \leftarrow V^{\text{c}} \cup \{u\}$;
return Q;

Algorithm 4.3 shows our approach for reducing the size of the open list. Concerning the overall efficiency, it is essential to be able to quickly identify valid and promising nodes for merging. The algorithm has a four-fold aim: (1) to preferably merge nodes that are

less likely part of a final longest path, (2) to only merge nodes associated with similar states, since this is likely to yield a strong relaxation, (3) to avoid merges of nodes that may lead to cycles in the DD, and (4) to ensure that the open list gets empty after a finite number of expansions. The last two aspects are crucial conditions to ensure a proper termination of the approach, and they are not trivially fulfilled due to the possibility to merge across different layers. The ways in which we handle these aspects require some problem-specific tailoring.

Aspect (1) is considered by iterating over the nodes in the open list in an increasing Z^{lp} order and trying to merge each node with a suitably selected other node in a pairwise fashion until the size of the open list does not exceed ϕ anymore. The motivation for the increasing Z^{lp} order is that A* search has so far postponed the expansion of these nodes while other nodes with comparable Z^{lp} values have already been expanded. Therefore, the nodes with small Z^{lp} values can be argued to appear less likely in a final longest path. We considered in preliminary experiments also an increasing f order, thus processing the priority queue essentially in reverse order. While we obtained mostly DDs of roughly comparable quality, they were sometimes significantly larger and more computation time was needed. The selection of the second node for merging is done considering aspects (2) to (4) by utilizing a global set of so-called *collector nodes* V^{c}. To this end, we define a problem-specific labeling function $L(u)$ that maps the data associated with a node u—in particular its state $\sigma(u)$—to a simpler label of a restricted finite domain $\mathcal{D}_L$. Consequently, the nodes are partitioned into subsets of similar nodes. For example, our labeling function may drop, aggregate, or relax parts of the states considered less important and condense the information in this way. Similar principles as in state-space relaxation [22] can be applied. The labeling function, however, may additionally also consider the upper bound $Z^{\text{ub}}(u)$ as criterion for similarity; experimental results in Section 4.12 will show the particular usefulness of this. The global set of collector nodes V^{c} is initially empty and is realized as a dictionary (e.g., hash table) indexed by the labels so that for each label in $\mathcal{D}_L$ there is at most one collector node in V^{c}, and thus $|V^{\text{c}}| \leq |\mathcal{D}_L|$. In this way, we can efficiently determine for any node u if a related collector node with the same label $L(u)$ already exists and, in this case, directly access it.

Algorithm 4.3 does this check for a current not yet expanded node u in Line 4. If the respective collector node v exists, is different from u, and also is not yet expanded, nodes u and v are merged, yielding the new node v' with state $\sigma(v') = \oplus(\{\sigma(u), \sigma(v)\})$. All incoming arcs from u and v will be redirected to the new node v'. Consequently, u is removed from Q and v from Q as well as V^{c}. Since we only merge unexpanded nodes, we do not have to consider any outgoing arcs. Next, we have to integrate the new node v' into the node set V. If there exists already a node v'' where the corresponding state $\sigma(v'')$ is equal to the merged state $\sigma(v')$ then we redirect all arcs from v' to v'' in order to avoid multiple nodes in set V associated with the same state. If the f value of node v'' got increased due to this arc redirecting, we insert node v'' into the open list Q to subsequently re-expand it with its successor states as needed. Otherwise, if such node v'' does not exist, we directly add node v' to set V and to the open list Q. In each case the

newly created node v' or the already existing node v''—if it is not yet expanded—shall become a collector node in V^{c}, essentially replacing the former collector node v. Node v' or v'' may, however, have a different label than the former v, and some other collector node with the same label as v' or v'' may already exist in V^{c}. In this case, we iterate the merging with these nodes by continuing the while-loop in Line 4. Algorithm 4.3 terminates when the open list has been reduced enough or—in the extreme case—after having unsuccessfully considered all nodes in Q for merging. The latter may happen only when all open nodes are either contained in V^{c} or are just scheduled for re-expansion. Note that the pseudo-code in Algorithm 4.3 shows only the main idea abstractly to point out the important steps. In a concrete implementation, a few additional corner cases need to be considered. In particular when collector nodes get changed (e.g., expanded) between two calls of Algorithm 4.3.

4.8.2 Problem-Specific Aspects

We now consider all further details to apply the general concept of the A*-based construction of relaxed DDs to PC-JSOCMSR. The most crucial problem-specific aspect is the labeling function which is employed for identifying nodes to merge. The labeling function not only has to avoid merges that may lead to cycles in the DD, but has also has a severe impact on the DDs's quality and size. We propose four labeling functions that combine different features of the state. Moreover, we discuss a merging strategy for dominated nodes that allows us in general obtaining substantial smaller DDs with only marginal impact on the DDs's longest paths. Finally, we elaborate the tie breaking criteria used in the open list's priority function.

Labeling Function for Collector Nodes

Here we define the labeling function $L(u)$ used for indexing the collector nodes V^{c}. Remember that this function shall partition the set of nodes into subsets. Nodes within a subset should preferably be considered similar enough to be promising to merge. Hence, similar nodes should receive the same label.

In case of PC-JSOCMSR, we use for a node u the triple $L(u) = (t_0(u), r(u), Z^{\mathrm{ub}}(u))$ as label, where $t_0(u)$ is again the time from which on the common resource is available, $r(u)$ refers to the secondary resource of the job scheduled last in the so far longest path to node u (ties are resolved by using the resource identified first), and $Z^{\mathrm{ub}}(u)$ is an upper bound for the still achievable path length from node u onward. Note that by this definition, we ignore the set of jobs that might still be scheduled $P(u)$. Moreover, we do not explicitly consider the individual availability times of the secondary resources $t_r(u)$, $r \in R$. Instead $r(u)$ is used as a rough substitute. The upper bound $Z^{\mathrm{ub}}(u)$ is an important additional indicator that summarize important information about the state of node u. In summary, two nodes may only be merged if (1) the common resource 0 is used to the same extent, (2) the last used secondary resource is the same, and (3) the values of the problem-specific upper bounds coincide.

When using this labeling function, two nodes $u, v \in V$ are only merged if $t_0(u) = t_0(v)$. Hence, the merged node will have the same t_0 value according to Equation (4.11). Since each job requires the common resource 0 for a positive time, scheduling a job increases the corresponding t_0 value. Consequently, each transition from a node to a successor node yields a state with a larger value for t_0. Furthermore, the compiled relaxed MDDs have the important property that the t_0 values strictly increase along any path in our MDD. Due to this property it holds that cycles cannot occur, the open list gets empty in a finite number of iterations, and Algorithm 4.2 terminates with a complete relaxed MDD. This is because the t_0 values strictly increase along any path and, thus, the set of jobs in P has to decrease due to the job's limited time windows.

In addition, we investigate in Section 4.12.3 further labeling functions which consist of different combinations of $t_0(u)$, $r(u)$ and $Z^{\text{ub}}(u)$: $L^1(u) = t_0(u)$, $L^2(u) = (t_0(u), r(u))$, and $L^3(u) = (t_0(u), Z^{\text{ub}}(u))$.

Besides the theoretical convergence, it might be the case that the practical running time of the algorithm is still too large due to the not strongly limited domain size of the labels: Values $t_0(u)$ as well as $Z^{\text{ub}}(u)$ are continuous and in the worst case exponentially many different values may emerge in the course of our algorithm, leading to a potentially exponential number of collector nodes. In our experiments in Section 4.12, this situation did not occur. In case that it does, it can be overcome by discretizing these values by dividing them with appropriate constants α_{t_0} and $\alpha_{Z^{\text{ub}}}$ and rounding down, yielding the discretized labeling function

$$L^{\text{adv}}(u) = \left(\lfloor t_0(u)/\alpha_{t_0} \rfloor, r(u), \lfloor Z^{\text{ub}}(u)/\alpha_{Z^{\text{ub}}} \rfloor \right). \tag{4.16}$$

To guarantee that cycles are avoided and finite convergence is achieved, it has to be ensured that $\alpha_{t_0} \leq \min_{j \in J} p_0$ holds in order that t_0 strictly increases on all paths in the MDD. Clearly, stronger discretizations, i.e., larger values for α_{t_0} and $\alpha_{Z^{\text{ub}}}$, will typically reduce the number of collector nodes maintained and thus lead to more merges and one can expect less overall nodes in the final relaxed MDD. However, the quality of the MDD and the upper bound obtained will in general also be weakened.

Dominated Merging

Algorithm 4.3 does not merge already expanded nodes since, in general, the operations of re-evaluating and updating the expanded sub-graphs would be too computationally expensive. However, under certain circumstances it is possible to merge nodes with already expanded collector nodes without further evaluations and updates. Let $v \in Q$ be a not yet expanded node and $u \in V$ be an already expanded node. If $t_0(v) = t_0(u)$, $Z^{\text{lp}}(v) \leq Z^{\text{lp}}(u)$, and $\oplus(\{\sigma(v), \sigma(u)\}) = \sigma(u)$ holds, then it is possible to merge v into u without changing the state of u or increasing the length of the MDDs's longest path. The first two conditions are used to safely omit the re-expansion of node u and to efficiently identify such possible merges by additionally indexing all so far encountered nodes $u \in V$ by their $t_0(u)$ values.

Each new or changed node in Q after a node expansion is considered for this type of merge. To this end, we check the condition with all other nodes in V that have the same t_0 value. If a pair of nodes $v \in Q$ and $u \in V$ that fulfills this condition is found, we remove v from the open list and merge v into u by redirecting all incoming arcs from v to u. Since this kind of merge does not introduce any relaxation loss, we perform this procedure after every node expansion even if $|Q| \leq \phi$.

Tie Breaking in the Priority Function

The nodes in the open list Q are sorted according to the value of the priority function f, given in Equality (4.15). It is typically the case that several nodes have the same f-values, and we therefore use the following two-stage tie breaking in order to further guide the algorithm promisingly. First, if two nodes have the same f-value, we always prefer exact nodes over non-exact nodes. We call a node exact when it has a longest path from the root node that does not contain any merged node where the merging gave rise to a relaxation loss. In other words, an exact node is guaranteed to have a feasible solution that corresponds to this longest path. Such nodes are considered more promising to expand than non-exact nodes with the same f-value. In case of a remaining tie, we prefer nodes where the corresponding state has fewer jobs that may still be scheduled, i.e., we prefer nodes u having a smaller cardinality of $P(u)$.

4.9 Filtering of Relaxed Decision Diagrams

Relaxed MDDs not only provide an upper bound on the optimal objective value of the original problem, but can be also exploited further. If we utilize a relaxed MDD to derive heuristic solutions as we will do in Section 4.10, it can make sense to further process it by *filtering* in order to reduce its size and to further strengthen it. In the following, we describe our approach to filter a relaxed MDD for the PC-JSOCMSR. The technique is related to the filtering in IRLP, described in Section 4.7.1.

The algorithm tries to remove arcs and nodes that cannot be part of a path representing an optimal solution. The purpose is to obtain a MDD with fewer arcs and nodes and at the same time strengthen the states of the remaining nodes, which frequently allows further reductions in turn. Because this filtering also removes arcs from paths that encode feasible but sub-optimal solutions, the resulting MDD is not a relaxation of the original exact MDD. It is, however, guaranteed that optimal solutions of the original problem are retained.

Our filtering is performed in a single top-down pass, i.e., by traversing the MDD with a breadth-first-search. For each considered node we remove incoming arcs that are only used by paths encoding sub-optimal solutions and outgoing arcs if all traversing paths correspond to infeasible solutions. Since the removal of an arc might enable the filtering of an already earlier processed arc, we reconsider the node's incident arcs until no further changes are achieved. Nodes that become unreachable from the root node $\mathbf{r}$, as they remain without any incoming arcs, are also removed together with all their outgoing arcs.

In more detail, an arc is removed if all **r**–**t** paths containing the arc are shorter than a given lower bound on the optimal objective value. This lower bound, denoted by Z^{lb}, can be either obtained from the longest path from **r** to an exact node in the given relaxed MDD or by a primal heuristic. A longest path traversing an arc $a = (u, v)$ is composed of the longest path from **r** to u, arc a, and the longest path from v to **t** and has length $Z^{\text{lp}}(u) + z(a) + Z^{\text{lp}\uparrow}(v)$, where $Z^{\text{lp}\uparrow}(v)$ is the longest path starting from node v. While $Z^{\text{lp}}(u)$ can be calculated and stored for each node $u \in V$ during the construction of the MDD, $Z^{\text{lp}\uparrow}(u)$ has to be still determined. This is done by an bottom-up breadth-first-search of the MDD, starting from **t** with $Z^{\text{lp}\uparrow}(\mathbf{t}) = 0$ and setting $Z^{\text{lp}\uparrow}(u) = \max\{z(a) + Z^{\text{lp}\uparrow}(v) \mid a = (u, v) \in A\}$ for any further encountered node u.

Remember that our A*-based compilation of relaxed MDDs also associates an upper bound $Z^{\text{ub}}(v)$ with each node $v \in V$. Sometimes this upper bound is tighter than the length of the longest v–**t** path in the current MDD, i.e., $Z^{\text{lp}\uparrow}(v)$, and thus, we should consider the smaller value. Furthermore, also the maximum Z^{ub} value of all predecessor nodes increased by the arc costs $z(a)$ as well as the maximum Z^{ub} value of all successor nodes decreased by $z(a)$ are respective bounds which might be stronger especially after some incident arcs have already been removed. Overall, we calculate the strongest bound

$$\widehat{Z}^{\text{lp}\uparrow}(v) = \min\left\{Z^{\text{lp}\uparrow}(v), Z^{\text{ub}}(v), \max_{a=(v,w)\in A}(Z^{\text{ub}}(w) + z(a)), \max_{a=(u,v)\in A}(Z^{\text{ub}}(u) - z(a))\right\}$$

and remove arc $a = (u, v)$ iff $Z^{\text{lp}}(u) + z(a) + \widehat{Z}^{\text{lp}\uparrow}(v) < Z^{\text{lb}}$ during the top-down traversal of the MDD.

The state associated with a node should always be the result of merging the states obtained by the respective transitions from the immediate predecessor nodes, followed by the strengthening described in Section 4.4. Consequently, whenever a node's incoming arc is removed, we also re-compute its associated state as well as the bound $Z^{\text{ub}}(v)$. In case of a change of the state we further check all outgoing arcs for their validity. Any arc that does not represent a feasible extension anymore is removed. Moreover, an updated state might induce changes in the successor states. Consequently, a node's state is re-evaluated if one of its predecessor states has been updated.

4.10 Accelerated Top-Down Construction of Restricted Decision Diagrams

A restricted MDDs represents only a subset of all feasible solutions, but no infeasible solutions. They are primarily used to obtain feasible heuristic solutions and corresponding lower bounds. The construction usually follows a layer-by-layer top-down approach [11, 13]. As for relaxed MDDs, the size of the obtained restricted MDDs is limited by a maximum allowed width β for each layer. Whenever the allowed width would be exceeded, nodes are selected from the current layer according to a greedy criterion and removed together with their incoming arcs. See Section 4.6 for further details on the TDC of restricted MDDs for PC-JSOCMSR.

So far, we are only aware of previous approaches that construct restricted MDDs independently of relaxed MDDs. However, an earlier construction of a relaxed MDD will, in general, have collected useful information about an instance of the problem and this is encoded in the relaxed MDD. Our proposed strategy for constructing a restricted MDD based on a relaxed one will exploit this information to potentially either speed up the construction of a restricted MDD and/or obtain a stronger restricted MDD representing better solutions. Our approach also applies the top-down compilation principle. The novel aspect is that we, during the construction, follow paths in the given relaxed MDD. We denote here all elements belonging to the restricted MDD with primed symbols, while the corresponding symbols of the relaxed MDD are not primed.

A node $u' \in V'$ in our restricted MDD always has a *corresponding node* $u \in V$ in the relaxed MDD M. More specifically, a path from $\mathbf{r}'$ to u' represents a feasible partial solution that is also represented in M by a path from $\mathbf{r}$ to u. In other words, the node $u' \in V'$ that corresponds to a node $u \in V$ is the node that can be reached by the same sequence of scheduled jobs. Considering this corresponding node u in the relaxed MDD allows us to skip certain transitions in the construction of the restricted MDD that otherwise would appear promising and for which arcs and new successor nodes would be created. In this way a vast amount of arcs and nodes for states that cannot lead to an optimal solution may be avoided.

Algorithm 4.4 shows our compilation of a restricted MDD in detail. The procedure takes the relaxed MDD G and a maximum allowed width β as input and returns the compiled restricted MDD G'. The compilation starts with the first layer that consists of the root node $\mathbf{r}'$, which is by definition the same as in the relaxed MDD. Then, each successive layer V'_{l+1} is built from the preceding layer V'_l by creating nodes and arcs for feasible transitions from the states associated with the nodes in V'_l. Here comes one of the novel aspects: For each node u' in layer V'_l we consider only state transitions corresponding to outgoing arcs of the respective node u in the relaxed MDD G. Other potentially feasible state transitions do not need to be considered since we know from the relaxed MDD that they cannot lead to an optimal feasible solution. Thus, we prevent the creation of arcs that have been filtered in the relaxed MDD by the filtering from Section 4.9. Note, however, that the relaxed node u might have outgoing arcs representing transitions that are actually infeasible for node u' in the restricted MDD. This may happen since the states of u' and u do not need to be the same but u' may also dominate u because some nodes have been merged in the path from $\mathbf{r}$ to u in the relaxed MDD. In Line 9, the algorithm therefore checks the feasibility of the respective transition (remember that $\hat{0}$ represents the infeasible state) and skips infeasible ones. For PC-JSOCMSR, this feasibility check simply corresponds to testing if $j(a) \in P(u)$. Moreover, when we have already reached the maximum allowed width at the current layer, we make an efficient pre-check if the node v' to be created next would be removed later when the set V'_{l+1} is greedily reduced to β nodes. To this end, we evaluate the criterion that is used to decide which nodes are removed from the current layer for the corresponding node v in the relaxed MDD in conjunction with the so far obtained set V'_{l+1}. If this criterion

Algorithm 4.4: Construction of a Restricted MDD Based on a Relaxed MDD

Input: relaxed MDD $G = (V, A)$, maximum width β
Output: restricted MDD $G' = (V', A')$

```
1  V'_1 ← {r'};
2  A' ← ∅;
3  l ← 1;
4  while V'_l ≠ ∅ do
5      V'_{l+1} ← {};
6      foreach node u' ∈ V'_l do
7          let u ∈ V be the node corresponding to u' w.r.t. the path from the root;
8          foreach outgoing arc a = (u, v) of node u do
9              if τ(σ(u'), j(a)) = 0̂ then
10                 continue with next arc;
11             if |V'_{l+1}| = β ∧ node v would be removed from V'_{l+1} ∪ {v} then
12                 continue with next arc;
13             Σ ← τ(σ(u'), j(a)); strengthen Σ;
14             if ∄v' ∈ V'_{l+1} | σ(v') = Σ then
15                 add new node v' to V'_{l+1} and set σ(v') = Σ;
16             add new arc a' = (u', v') to A';
17             if |V'_{l+1}| > β then
18                 select and remove a node from V'_{l+1} with its incoming arcs
                     according to a greedy criterion;
19     l ← l + 1;
20 return G' = (V', A') with V' = V'_1 ∪ ... ∪ V'_{l-1};
```

is chosen in a sensible way, the evaluation for v will never indicate a removal of node v when v' would not be removed, since either the associated states are identical or the state of v' dominates the state of v. In our algorithm, Line 11 realizes this pre-check and correspondingly skips the respective transitions. For the remaining transitions, Line 13 calculates the obtained new state Σ and creates the corresponding node v' if no node in V' exists yet for Σ. Then, an arc (u', v') representing the transition in the restricted MDD is created and added to A'. Finally, if V'_{l+1} has grown to $\beta + 1$ nodes, a node is removed according to the used greedy criterion.

After having compiled the restricted MDD, a best feasible solution is given by a longest path in it, which can be directly obtained through the Z^{lp} values again calculated for all nodes. The quality and the time required to build the restricted MDDs with Algorithm 4.4 is mainly influenced by the maximum allowed width β, the given relaxed MDD G, and the criterion to select nodes for removal. As usual, restricted MDDs of larger width require more time to be compiled but encode more feasible solutions and can thus be

expected to get closer to an optimum. The benefits of exploiting a relaxed MDD in the compilation of a restricted MDD depends on how closely the exact states in the restricted MDD are approximated by the corresponding states in the relaxed MDD as well as on the effectiveness of the relaxed MDD's filtering.

A typical way to select the nodes for removal at each layer is to take the nodes with the smallest lengths of the longest paths to reach them from the root node $\mathbf{r}'$, i.e., the nodes with the smallest $Z^{\text{lp}}(v')$, $v' \in V'_{l+1}$ [11, 13]. As already pointed out in Section 4.6 this strategy is not beneficial for PC-JSOCMSR since it disregards the advances in the time line. Instead, we remove nodes with the smallest $Z^{\text{lp}}(v')/t_0(v')$ ratios in our implementation for the PC-JSOCMSR as it was already done in Section 4.6.

When applying this removal criterion to the corresponding node v of the relaxed MDD in Line 11, it holds that $Z^{\text{lp}}(v)/t_0(v) \geq Z^{\text{lp}}(v')/t_0(v')$ as $Z^{\text{lp}}(v) \geq Z^{\text{lp}}(v')$ and $t_0(v) \leq t_0(v')$ as state $\sigma(v')$ is equal to or dominates state $\sigma(v)$. We can even sharpen this estimation by using $(Z^{\text{lp}}(u') + z(a))/t_0(v)$ and thus take advantage of our knowledge of $Z^{\text{lp}}(u')$ and the respective transition costs $z(a)$ to reach node v.

Another reasonable option would be to select the nodes for removal with the smallest $Z^{\text{lp}}(v') + \widehat{Z}^{\text{lp}\uparrow}(v)$ values. Observe, that we reuse here the upper bounds $\widehat{Z}^{\text{lp}\uparrow}(v)$ computed during the filtering of the relaxed MDD. Similarly, we can use as before the possibly stronger bound $Z^{\text{lp}}(u') + z(a) + \widehat{Z}^{\text{lp}\uparrow}(v)$ in Line 11 to pre-check a transition before it is exactly calculated. Preliminary tests, however, have shown that this node removal strategy is rather sensitive to effects caused by merged nodes in the relaxed MDD and, thus, is not effective. We therefore did not pursue it further.

4.11 General Variable Neighborhood Search

In this section the GVNS is presented which serves us as a reference approach for obtaining heuristic solutions. The GVNS metaheuristic [47] is a prominent local search based metaheuristic which operates on multiple neighborhoods. To recapitulate Section 2.3.5 the central idea is to systematically change local search neighborhood structures until a local optimum in respect to all these neighborhood structures is found. This part is called *variable neighborhood descent* (VND). To further diversify the search, the GVNS performs so-called shaking for local optimal solutions by applying random moves in larger neighborhoods. These perturbed solutions then undergo VND again, and the whole process is repeated until a termination condition is met at which point the best solution encountered is returned.

In the context of this metaheuristic we represent a solution by a permutation $\pi = (\pi_i)_{i=1,\ldots,|J|}$ of the entire set of jobs J. Starting times and the subset of jobs $S \subseteq J$ that actually is scheduled is obtained by considering all jobs in the order of π and determining each job's earliest feasible time; jobs that cannot be feasibly scheduled w.r.t. their time windows anymore are skipped. This solution representation allows us to use rather simple neighborhood structures.

Our GVNS for PC-JSOCMSR starts with a random permutation of the jobs J as initial solution. In a preliminary study, we also used initial solutions computed by a FRB4$_k$ [99] construction heuristic (see also Section 3.8.1). Although this constructive heuristic provided much better starting solutions, we could not observe significant differences in the quality of final solutions returned by the GVNS.

We employ in our GVNS two local search neighborhood structures. The first one considers all exchanges of pairs of jobs within the permutation, while the second considers the removal of any single job and its re-insertion at another position. To avoid the consideration of moves that do not change the actual schedule, we require that each move changes either S or the order of the jobs within S.

In the VND, we apply any possible improving exchange move before considering the moves that remove and reinsert jobs. Each neighborhood is searched in a first improvement fashion. As shaking operation, we perform a sequence of k random remove-and-insert moves. Whenever a new incumbent local optimal solution is found, the following shaking starts with $k = 1$. Parameter k is increased by one up to a maximum value $k_{\max}$ after every unsuccessful shaking followed by the VND. After reaching $k_{\max}$, k is reset to one again.

4.12 Computational Study

In this section we report results from the experimental evaluation of the approaches proposed in this chapter. The algorithms are implemented in C++ and have been compiled by GNU G++ 7.3.1. All experiments are performed on a single core of an Intel Xeon E5-2640 v4 CPU with 2.40GHz and 16 GB of memory. Throughout this section we denote by TDC the classic top-down construction for relaxed and reduced MDDs as described in Section 4.6. We refer by IRLP to our incremental refinement compilation of relaxed MDDs for PC-JSOCMSR from Section 4.7. The A*-based construction of relaxed MDDs is abbreviated in the following by A^*C. If we derive restricted MDDs by the construction method presented in Section 4.10 from relaxed MDDs provided by A^*C, we write A^*C+TDC. As we will argue later, we always perform the filtering from Section 4.9 within A^*C+TDC. Finally, we denote the metaheuristic from Section 4.11 that serves us as a reference approach simply by GVNS.

We start in the following section with a description of the used benchmark instances. In Section 4.12.2 our initial layer-based compilation methods are highlighted. To this end, we first compare the relaxed MDDs build by the TDC and the IRLP followed by assessing the quality of the restricted MDDs compiled by TDC in comparison to the solutions obtained by the GVNS. Then, we shift our focus towards A^*C. Section 4.12.3 presents results from studying the impact of different values for the open list size limit ϕ and of different choices for the labeling function $L(u)$ in the compilation of relaxed MDDs with A*C. Afterwards, in Section 4.12.4, we compare the quality of upper bounds obtained from A*C to those from other approaches. The impact of filtering relaxed MDDs is addressed in Section 4.12.5, and later Section 4.12.4 compares primal bounds

obtained by A^*C+TDC to those from the other presented approaches and a reference MILP and CP model.

4.12.1 Benchmark Instances

We have generated three non-trivial sets of benchmark instances that are available at https://www.ac.tuwien.ac.at/research/problem-instances. In the first series of experiments, discussed in Section 4.12.2, we use two sets of benchmark instances based on characteristics from the particle therapy application. Each of these two sets contain in total 840 instances with 30 instances for each combination of $n \in \{10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 120, 150, 200, 300\}$ jobs and $m \in \{2, 3\}$ secondary resources. The instances allow that roughly 30% of all the jobs can be scheduled. These benchmark sets, called P^{B} and P^{S}, differ in the usage of the secondary resources. While the instances of set P^{B} have a *balanced* workload w.r.t. the secondary resources, the instances of set P^{S} have a *skewed* workload, meaning that one secondary resource is required substantially more often than the other secondary resources. This workload balance played in Horn et al. [55] an important role for the algorithms to solve the JSOCMSR problem in which all jobs have to be scheduled. It turned out that the impact in the PC-JSOCMSR problem variant, however, is typically less relevant. This follows from the fact that in PC-JSOCMSR only a subset of all jobs needs to be selected and scheduled, which mitigates a resource imbalance. Therefore, we consider in the later experiments concerning the A^*C just the balanced instance set P^{B} and extend it by instances with up to 500 jobs. Furthermore, we introduce an additional instance set A for the avionic system scheduling scenario, which has substantially different characteristics than instance set P^{B}. To summarize, from Section 4.12.3 on, we consider instance set P^{B} and A that contain 30 instances for each combination of $n \in \{50, 100, \ldots, 500\}$ jobs and $m \in \{2, 3\}$ secondary resources for instance set P^{B} and $m \in \{3, 4\}$ secondary resources for instance set A.

In the benchmark instances of type P^{B} the secondary resources are equally distributed among the jobs and each job requires in expectation the common resource for the second third of its processing time. To this end a job's secondary resource is uniformly sampled from R. The processing time of a job j is determined by sampling values for p_j^{pre}, p_j^{post} from $\mathcal{U}\{0, 8\}$ and for p_j^0 from $\mathcal{U}\{1, 8\}$. In the set of instances P^{S} one of the secondary resources is required predominantly. Moreover, in expectation the common resource is required more than the half the job's processing time. In detail, a job's secondary resource is set to m with a probability of 0.5 while the other resources in R are selected with a probability of $1/(2m-2)$. The duration p_j^0 of the jobs $j \in J$ are chosen uniformly from $\{1, \ldots, 13\}$ and the pre-processing and post-processing times p_j^{pre} and p_j^{post} are both uniformly selected from $\{0, \ldots, 5\}$. The remaining characteristics of the two benchmark sets are obtained in the same way: The prize z_j associated with each job is sampled uniformly from $\{p_j^0, \ldots, 2p_j^0\}$ in order to correlate with the time the common resource is used. For the jobs we generate between one and three time windows in such a way that approximately 30% of the jobs fit into a schedule. To this end, we sample for each job the number of time windows ω_j from $\{1, 2, 3\}$. Moreover, let $\mathrm{E}(\mathrm{p}^0)$ be the expected duration a job requires

the common resource and let $T = \lfloor 0.3\, n\, \mathrm{E}(\mathrm{p}^0) \rfloor$ be the total expected time required from the common resource to schedule 30% of all jobs. The ω_j time windows W_j for job j are generated as follows: We choose a start time W_{jw}^{start} uniformly from $\{0, \ldots, T - p_j\}$ and an end time W_{jw}^{end} from $\{W_{jw}^{\text{start}} + \max(p_j, \lfloor 0.1\, T/\omega_j \rfloor), \ldots, W_{jw}^{\text{start}} + \max(p_j, \lfloor 0.4\, T/\omega_j \rfloor)\}$. If we obtain overlapping time windows, they are merged and ω_j is adjusted accordingly.

Instance set A is based on the avionics scenario (see Section 4.3.2). The instances have a fixed time horizon $T = 1000$ and the set of all considered jobs is partitioned into 20% communication jobs, 40% partition jobs and 40% regular jobs. Time p_j^0 is sampled for partition jobs and regular jobs from the discrete uniform distribution $\mathcal{U}\{36, 44\}$. For communication jobs the time the common resource is needed is set to the fixed value 40. Each partition job is assigned a secondary resource uniformly selected from $\{1, \ldots, m-1\}$. The total processing time p_j is sampled for partition jobs from $\mathcal{U}\{5p_j^0, 8p_j^0\}$ and then, with equal probability, p_j^{pre} or p_j^{post} is set to 0 and the respective other value is set to $p_j - p_j^0$. Since the communication jobs and regular jobs do not use a secondary resource in the real scenario, an artificial secondary resource is introduced and assigned to all of these jobs. More specifically, both the communication jobs and the regular jobs require resource $m \in R$ for $p_j = p_j^0$. In addition, this artificial secondary resource means that the number of secondary resources for an instance is always one more than the number of application modules in the system. The prize z_j is for five of the partition jobs and ten of the communication jobs set to the value 70 to give these jobs a higher priority, while for the remaining partition jobs and communication jobs the prize is sampled from $\mathcal{U}\{10, 50\}$. For all regular jobs, the prize is sampled from $\mathcal{U}\{10, 25\}$. For partition jobs and regular jobs, the number of time windows and the length of the time windows are computed as for the instance sets P^{B} and P^{S}, but for the communication jobs the structure is rather different. Communication jobs can only be scheduled at certain points in time when the communication can be performed, and in our case these time points are at $0, 80, 160, \ldots, 880$. Each time window of a communication job starts at one of such time points and has a length equal to the job's processing time. Moreover, the communication job's time windows always start at a consecutive sequence of these time points. The number of time windows for a communication job is obtained by sampling a value from the uniform distribution $\mathcal{U}\{1, 3\}$ and multiply it by three.

Note that the instances of sets P^{B} and A differ substantially in their structure. For instances of set P^{B}, the number of jobs that may be feasible scheduled grows linearly with the instance size, i.e., the number of jobs. Moreover, the prize for each job is correlated to the time the job needs the common resource. In contrast, for instances of set A the number of jobs that may be feasible scheduled stays in the same order of magnitude due to the fixed time horizon of $T = 1000$. Furthermore, the prize structure is very different since the prize does not depend on the jobs' processing time but instead on a priority. As we will see in the results, this structural differences also impact the obtained relaxed MDDs. In particular, the height of relaxed MDDs compiled for instances of set P^{B} grows with the problem size, whereas the relaxed MDDs obtained for instances of set A typically have a comparable height.

4.12.2 Top Down Construction and Incremental Refinement

This section first compares the relaxed MDDs compiled by the TDC and the IRLP for PC-JSOCMSR. Afterwards, the restricted MDDs constructed by the TDC are set into perspective to the results obtained from the GVNS. We start by giving details of the experimental setting.

The initial relaxed MDD used by incremental refinement methods in the literature [23, 68] are typically trivial ones of width one and can be obtained by calling TDC with $\beta = 1$. For PC-JSOCMSR there is a more meaningful initial relaxed MDD of maximum width m, where on each layer all states are merged that are obtained by jobs requiring the same secondary resource. This initial relaxed MDD has in general already significantly stronger states than the relaxed MDD of width one, because in the latter the advances on the times t_r for the secondary resources $r \in R$ cancel each other out. Preliminary experiments showed that small instances can be optimally solved with fewer iterations and on larger instances stronger bounds can be obtained when providing the initial relaxed MDD of width m. Hence, we do this in all our further IRLP runs.

In other preliminary experiments we investigated different configurations of the GVNS. We tried changing the order of the neighborhood structures within the VND and also shaking operators based on exchanging the positions of randomly selected jobs. The configuration described in Section 4.11 was found to work best. Moreover, we tuned the maximum shaking size parameter $k_{\max}$. Relatively small values for $k_{\max}$ turned out to typically yield better results, and we decided to use $k_{\max} = 4$ for all further experiments.

In the first series of experiments we compare the quality of relaxed MDDs compiled by TDC and IRLP, respectively. IRLP was performed with a CPU-time limit of 900 seconds per run, while for TDC we used different values for the maximum width β in dependence of the number of jobs so that the required CPU-time was in a similar order of magnitude. Table 4.1 and Table 4.2 shows average results for P^B and P^S instances, respectively. The first three columns describe the instance properties. For both approaches mean dual bounds $\overline{Z^{\text{lp}}}$ are listed together with the corresponding standard deviations $\sigma(Z^{\text{lp}})$, the median numbers of nodes of the relaxed MDD $|V|$ and median completion times t in seconds. In addition, the employed maximum width β in the TDC are given. Moreover, for the IRLP we state in column $\Delta[\%]$ the percentage of nodes saved due to the elimination of duplicate states. In other words, we compute $100 - 100 \cdot (|V|/|V'|)$, where $|V'|$ is the size of the MDDs of separate runs of the IRLP without elimination of duplicate nodes, as it was presented in our preliminary work [79].

On the smallest instances both algorithms produce relaxed MDDs with the same dual bounds. In these cases the obtained bounds correspond to the optimal objective values, which we verified by checking that the longest paths indeed correspond to feasible schedules. In fact, TDC could solve several instances with up to 60 jobs, while IRLP found optimal solution for some instances with up to 50 jobs. While on the medium to large instances with balanced jobs we cannot observe a clear tendency which method provides tighter bounds, IRLP outperforms TDC by a rather large margin on almost all

			TDC (relaxed MDDs)					IRLP				
type	m	n	β	$\|V\|$	$t[s]$	$\overline{Z^{\text{lp}}}$	$\sigma(Z^{\text{lp}})$	$\|V\|$	$\Delta[\%]$	$t[s]$	$\overline{Z^{\text{lp}}}$	$\sigma(Z^{\text{lp}})$
P^B	2	010	750000	42	<1	**30.93**	6.91	20	11	<1	**30.93**	6.91
P^B	2	020	750000	526	<1	**50.37**	5.71	182	14	<1	**50.37**	5.71
P^B	2	030	750000	5365	<1	**75.33**	6.41	1258	17	<1	**75.33**	6.41
P^B	2	040	750000	78336	<1	**98.93**	7.05	12618	18	2	**98.93**	7.05
P^B	2	050	750000	850366	18	**123.83**	9.86	210294	34	374	125.80	10.93
P^B	2	060	750000	6192518	214	181.07	26.79	1502938	36	900	**175.13**	14.89
P^B	2	070	100000	1821669	136	314.30	30.97	4160276	33	900	**253.23**	31.06
P^B	2	080	100000	2291714	243	400.57	35.81	4449456	26	900	**342.70**	48.38
P^B	2	090	100000	3109741	439	497.17	51.51	3722248	27	900	**454.70**	66.55
P^B	2	100	100000	3885520	683	605.10	47.07	3143184	29	900	**593.33**	100.04
P^B	2	120	20000	1096548	279	**868.50**	85.35	2581056	19	900	872.83	148.05
P^B	2	150	20000	1678748	690	**1245.50**	99.96	1867372	14	900	1409.60	148.46
P^B	2	200	2000	289016	232	**2176.47**	206.23	1300659	8	900	2283.80	117.48
P^B	2	300	2000	512774	974	**3830.17**	291.98	619908	5	900	3865.90	99.82
P^B	3	010	750000	52	<1	**36.17**	6.22	30	5	<1	**36.17**	6.22
P^B	3	020	750000	978	<1	**59.27**	7.85	336	12	<1	**59.27**	7.85
P^B	3	030	750000	13766	<1	**86.30**	7.08	4046	23	<1	**86.30**	7.08
P^B	3	040	750000	215763	3	**112.00**	7.79	68336	25	55	112.97	9.82
P^B	3	050	750000	3893395	84	**154.43**	24.57	718030	32	900	160.80	17.61
P^B	3	060	750000	10316441	474	241.53	16.68	3910070	21	900	**221.60**	15.19
P^B	3	070	100000	2441857	193	405.50	51.30	4870403	22	900	**334.90**	47.25
P^B	3	080	100000	3282533	355	527.47	56.95	3808838	24	900	**477.07**	60.86
P^B	3	090	100000	4259832	664	**655.80**	68.22	3207024	27	900	672.37	79.23
P^B	3	100	100000	5214238	981	**783.30**	76.89	3394374	19	900	840.93	67.22
P^B	3	120	20000	1552652	402	**1176.57**	91.67	2249186	23	900	1186.10	73.63
P^B	3	150	20000	2290835	1000	1687.27	137.87	1750716	11	900	**1619.23**	96.31
P^B	3	200	2000	381135	294	2827.77	161.11	1192373	2	900	**2364.27**	131.87
P^B	3	300	2000	598301	1318	4562.17	122.92	536464	14	900	**3914.33**	127.36

Table 4.1: Comparison of the relaxed MDDs obtained from TDC and IRLP on instance set P^B.

skewed instances. Notice that the size of the relaxed MDDs produced by both algorithms peaks by instances with 60 or 70 jobs and declines for larger benchmark instances. This can be explained for the TDC by the increasing number of state transitions that have to be performed for each layer and by the increasing number of nodes that have to be merged as a result. For IRLP the reason is similar. IRLP has to reevaluate for larger instances more frequently nodes with many incoming arcs.

IRLP has been extended w.r.t. our preliminary work [79] by the elimination of duplicate states within layers. While the effects on the obtained dual bounds have been negligible, the size of the produced MDDs has been reduced between by 10% to 30% on average. Smaller MDDs of similar quality become especially important if the MDDs are further exploited within a larger algorithmic framework such as in the DD-based branch-and-bound [13] or for constraint propagation in CP [23, 68].

In a second series of experiments the heuristic solutions obtained by the TDC for restricted

			TDC (relaxed MDDs)					IRLP				
type	m	n	β	$\|V\|$	$t[s]$	$\overline{Z^{\text{lp}}}$	$\sigma(Z^{\text{lp}})$	$\|V\|$	$\Delta[\%]$	$t[s]$	$\overline{Z^{\text{lp}}}$	$\sigma(Z^{\text{lp}})$
P^S	2	010	450000	40	<1	**50.93**	8.36	21	11	<1	**50.93**	8.36
P^S	2	020	450000	1039	<1	**89.93**	8.23	446	9	<1	**89.93**	8.23
P^S	2	030	450000	21220	<1	**131.37**	10.37	9478	26	1	**131.37**	10.37
P^S	2	040	450000	430093	7	**180.07**	12.40	167558	25	598	186.27	16.49
P^S	2	050	450000	4394388	128	300.13	50.61	2079225	25	900	**297.37**	30.72
P^S	2	060	450000	8549486	530	535.90	77.47	6311374	26	900	**463.60**	44.08
P^S	2	070	100000	3230809	321	835.43	119.97	5264954	29	900	**717.07**	74.14
P^S	2	080	100000	4362590	546	1091.63	124.31	4373746	25	900	**931.40**	94.35
P^S	2	090	100000	5475532	893	1315.33	120.45	4196128	27	900	**1154.03**	100.01
P^S	2	100	20000	1439179	287	1754.63	163.18	3287628	19	900	**1461.50**	102.69
P^S	2	120	20000	1840614	537	2276.60	236.54	2713622	15	900	**1891.17**	132.51
P^S	2	150	20000	2756871	1218	3315.60	209.25	1929511	13	900	**2598.47**	152.95
P^S	2	200	1000	199201	180	4853.90	171.69	1203598	9	900	**3770.27**	181.75
P^S	2	300	1000	299301	791	7483.10	187.80	669654	6	900	**6249.60**	213.77
P^S	3	010	450000	46	<1	**51.97**	9.76	34	7	<1	**51.97**	9.76
P^S	3	020	450000	1216	<1	**96.47**	9.13	470	20	<1	**96.47**	9.13
P^S	3	030	450000	23358	<1	**135.90**	9.42	9650	25	1	**135.90**	9.42
P^S	3	040	450000	1099240	15	**191.20**	17.19	377388	31	900	206.27	20.87
P^S	3	050	450000	5968862	211	357.60	57.78	5591745	31	900	**341.23**	36.53
P^S	3	060	450000	11241455	663	610.37	70.78	6410652	29	900	**548.23**	74.03
P^S	3	070	100000	4134692	401	956.73	114.00	5087212	29	900	**779.00**	73.10
P^S	3	080	100000	4676286	624	1219.10	166.32	4280442	24	900	**1012.07**	70.41
P^S	3	090	100000	6803302	1145	1623.87	162.81	3768948	23	900	**1249.00**	100.66
P^S	3	100	20000	1691990	313	2013.37	239.78	3123381	22	900	**1489.03**	128.71
P^S	3	120	20000	2298596	648	2696.10	208.48	2441742	15	900	**1926.13**	153.86
P^S	3	150	20000	2973857	1456	3510.93	225.37	1806430	12	900	**2669.80**	99.54
P^S	3	200	1000	199201	208	4904.30	165.27	1132778	9	900	**3901.67**	194.42
P^S	3	300	1000	299301	800	7508.93	188.07	528610	2	900	**6393.83**	250.52

Table 4.2: Comparison of the relaxed MDDs obtained from TDC and IRLP on instance set P^S.

MDDs are compared with the ones computed by the GVNS. We employ for the GVNS a time limit of 900 CPU-seconds as termination criterion. For TDC, different maximum widths β were used again so that the running times are in a similar order of magnitude. Table 4.3 and Table 4.4 shows the obtained results. The first three columns describe the instance properties and each row shows average results of 30 corresponding instances. The means and the corresponding standard deviations of the final objective values for TDC and GVNS are shown in the columns $\overline{Z^{\text{lp}}}$, $\sigma(Z^{\text{lp}})$, $\overline{obj}$ and $\sigma(obj)$, respectively. In addition, for TDC the maximum width β, median number of nodes in the restricted MDD $|V|$, and median computation times t in seconds are listed. Moreover, column t^{best} shows for the GVNS median times in seconds when the best solution has been found.

The TDC for restricted MDDs is able to outperform the GVNS on most of our benchmark instances. Only for the largest instances with three secondary resources or skewed jobs, GVNS is able to provide better results. The main reason for the superior performance

			TDC (restricted MDDs)					GVNS		
type	m	n	β	$\|V\|$	$t[s]$	$\overline{Z^{\text{lp}}}$	$\sigma(Z^{\text{lp}})$	$\overline{obj}$	$\sigma(obj)$	$t^{\text{best}}[s]$
P^B	2	010	750000	42	<1	**30.93**	6.91	**30.93**	6.91	<1
P^B	2	020	750000	526	<1	**50.37**	5.71	**50.37**	5.71	<1
P^B	2	030	750000	5365	<1	**75.33**	6.41	**75.33**	6.41	<1
P^B	2	040	750000	78336	1	**98.93**	7.05	98.90	7.01	<1
P^B	2	050	750000	850366	23	**123.27**	10.33	123.20	10.34	<1
P^B	2	060	750000	6149522	212	**146.80**	10.39	146.53	10.38	<1
P^B	2	070	750000	10039410	697	**172.23**	11.17	171.93	11.41	8
P^B	2	080	150000	2678998	259	**199.97**	13.36	199.47	13.13	8
P^B	2	090	150000	3275627	424	**231.83**	13.36	230.70	13.73	31
P^B	2	100	150000	3756246	610	**260.40**	11.52	258.83	11.71	67
P^B	2	120	50000	1592216	371	**315.90**	13.05	312.80	12.44	186
P^B	2	150	50000	2132142	799	**402.43**	18.65	396.50	17.47	551
P^B	2	200	6000	353180	226	**528.17**	18.96	526.83	18.73	459
P^B	2	300	6000	524990	766	**796.17**	16.68	791.67	17.26	751
P^B	3	010	750000	52	<1	**36.17**	6.22	**36.17**	6.22	<1
P^B	3	020	750000	978	<1	**59.27**	7.85	**59.27**	7.85	<1
P^B	3	030	750000	13766	<1	**86.30**	7.08	**86.30**	7.08	<1
P^B	3	040	750000	215763	8	**112.00**	7.79	111.93	7.85	<1
P^B	3	050	750000	3891532	105	**140.33**	10.40	140.27	10.40	1
P^B	3	060	750000	9554024	414	**165.13**	8.83	164.80	8.80	7
P^B	3	070	750000	13161224	1045	**194.83**	11.13	194.17	11.26	44
P^B	3	080	150000	3626754	354	**227.87**	13.54	226.70	13.34	72
P^B	3	090	150000	4210254	585	**257.53**	8.67	255.27	9.14	55
P^B	3	100	150000	4902283	822	**290.53**	14.30	288.03	14.60	274
P^B	3	120	50000	1996235	459	**352.27**	15.12	348.63	14.12	255
P^B	3	150	50000	2489353	936	**439.87**	15.88	436.30	16.68	358
P^B	3	200	6000	408372	245	579.97	18.74	**583.80**	16.31	507
P^B	3	300	6000	606960	830	847.67	16.25	**865.20**	19.25	707

Table 4.3: Comparison of heuristic solutions for P^B instances obtained from restriced MDDs compiled by TDC and the GVNS.

of the TDC on instances with balanced jobs and two secondary resources is that the corresponding exact MDDs are much smaller compared with the other instances. This can be seen on the smallest instances where the imposed maximum width is not yet restrictive. On the instances with 30 jobs, for example, the resulting MDDs for balanced jobs with two secondary resources have on average 5365 nodes, with three secondary resources 13766 nodes, and for the instances with skewed jobs there are 21220 and 23358 nodes. It is safe to assume that this difference in size becomes even larger with more jobs. To stay within the memory and time limits, the maximum allowed width has to be decreased with the increasing number of jobs, which becomes more and more restrictive

			TDC (restricted MDDs)					GVNS		
type	m	n	β	$\|V\|$	$t[s]$	$\overline{Z^{\text{lp}}}$	$\sigma(Z^{\text{lp}})$	$\overline{obj}$	$\sigma(obj)$	$t^{\text{best}}[s]$
P^S	2	010	750000	40	<1	**50.93**	8.36	**50.93**	8.36	<1
P^S	2	020	750000	1039	<1	**89.93**	8.23	**89.93**	8.23	<1
P^S	2	030	750000	21220	<1	**131.37**	10.37	**131.37**	10.37	<1
P^S	2	040	750000	430093	10	**180.07**	12.40	180.00	12.38	<1
P^S	2	050	750000	6335677	175	**225.67**	12.77	225.40	12.80	1
P^S	2	060	750000	10873978	679	**277.53**	11.67	276.87	11.97	10
P^S	2	070	150000	3233630	293	**326.20**	14.24	325.20	14.51	42
P^S	2	080	150000	3959832	452	**375.80**	16.50	374.23	16.66	70
P^S	2	090	150000	4495445	655	**421.50**	17.43	419.27	17.37	159
P^S	2	100	150000	5105289	962	**479.13**	20.91	476.03	20.84	259
P^S	2	120	50000	2072133	525	**574.37**	21.79	570.70	22.32	239
P^S	2	150	50000	2741184	1062	715.93	14.22	**716.37**	16.28	401
P^S	2	200	6000	432746	270	931.57	21.90	**948.87**	22.25	632
P^S	2	300	6000	668570	938	1382.70	30.42	**1424.07**	38.56	784
P^S	3	010	750000	46	<1	**51.97**	9.76	**51.97**	9.76	<1
P^S	3	020	750000	1216	<1	**96.47**	9.13	**96.47**	9.13	<1
P^S	3	030	750000	23358	<1	**135.90**	9.42	**135.90**	9.42	<1
P^S	3	040	750000	1099240	31	**185.43**	10.92	**185.43**	10.92	<1
P^S	3	050	750000	8572086	298	**234.40**	10.92	234.17	11.10	9
P^S	3	060	750000	13723665	842	**286.97**	13.00	286.10	13.13	9
P^S	3	070	150000	3656983	326	**331.30**	17.37	330.20	17.71	51
P^S	3	080	150000	4253940	475	**384.77**	17.27	383.33	17.38	37
P^S	3	090	150000	5157731	754	**429.60**	16.92	427.97	16.93	119
P^S	3	100	150000	5794926	1035	**487.30**	19.41	486.00	18.28	163
P^S	3	120	50000	2308306	575	565.13	17.12	**568.63**	16.74	448
P^S	3	150	50000	2904060	1060	708.07	21.61	**716.00**	20.42	442
P^S	3	200	6000	460541	267	928.80	24.19	**961.70**	24.27	519
P^S	3	300	6000	715920	926	1378.53	38.42	**1428.57**	39.68	747

Table 4.4: Comparison of heuristic solutions for P^S instances obtained from restriced MDDs compiled by TDC and the GVNS.

for the largest instances. Note that this relation can also be observed in Table 4.1 and Table 4.2 for relaxed MDDs. The GVNS approach, on the other hand, seems to be less affected by the instance type or by the number of secondary resources. This can be seen by the times the GVNS finds the final solution, which increases with the instance size but does not change substantially with the instance properties.

Concerning the gaps between the upper bounds obtained from the relaxed MDDs and the lower bounds from the heuristic solutions (compare Tables 4.1 and 4.3 or Tables 4.2 and 4.4), we can observe that they are only small for the small and medium-sized instances but become rather large for our largest instances. For example for the skewed instances

with 300 jobs, this gap even exceeds 340%. This also somewhat illustrates the difficulty of the considered problem and the limits of MDDs – or at least the limits of the considered construction methods.

4.12.3 Effects of Open List Size Limit and Labeling Functions

We tested A*C with different open list size limits ϕ and four different variants of the labeling function $L(u)$ used for mapping nodes to collector nodes. The considered labeling function variants, developed in Section 4.8.2, are $L^1(u) = t_0(u)$, $L^2(u) = (t_0(u), r(u))$, $L^3(u) = (t_0(u), Z^{\text{ub}}(u))$ and $L^4(u) = L(u) = (t_0(u), r(u), Z^{\text{ub}}(u))$. Figure 4.4 and Figure 4.5 document the effects of different values for ϕ and the choice of the labeling function on the instances with 250 jobs of set P^{B} with $m \in \{2, 3\}$ and of set A with $m \in \{3, 4\}$, respectively.

For each combination of a value for ϕ and a labeling function, a box plot summarizes the obtained results over all 30 instances with the same instance properties. The diagrams in the first row show the lengths of the longest paths from the obtained relaxed MDDs. The diagrams in the second row depict the measured CPU-times needed for compiling the MDDs. Moreover, in the last row the diagrams state the total number of nodes in the final relaxed MDDs.

In Figure 4.4 that considers the P^{B} instances, results are shown for $\phi \in \{1000, 2000, 3000, 5000\}$. As one could expect, we see that with increasing ϕ the lengths of the longest paths of the obtained relaxed MDDs get in general smaller, i.e., the obtained upper bounds become stronger. The MDD sizes and computation times on the other hand increase. Thus, parameter ϕ indeed allows to control the MDD's size, although not in such a direct linear way as the width-limit in a classical TDC. Furthermore, this effect can be observed for all labeling functions. Concerning the different labeling functions, $L^1(u) = t_0(u)$ yields relaxed MDDs with typically the smallest size, but also the weakest upper bounds. Additionally, this is achieved within the shortest computation times. The reason for this behavior is that labeling function L^1 does only consider the time from which on the common resource is available and has therefore the smallest domain $\mathcal{D}_{L^1}$ among the four considered labeling functions. Hence, the use of labeling function L^1 causes in general more node merges compared to the other more complex labeling functions which have larger domains and consequently also provide a finer differentiation of nodes. The additional consideration of $r(u)$ or $Z^{\text{ub}}(u)$ in the labeling function improves the obtained upper bounds in general substantially. Furthermore, the combination of all these aspects in L^4 provides the best results, this, however, at the costs of larger MDDs and higher running times. The smallest median longest path lengths of 783 and 808 for instances with two and three secondary resources, respectively, were obtained when limiting the size of the open list to $\phi = 5000$ nodes and using L^4. Notice further that parameter ϕ has more impact when using labeling function L^1 and less influence when using labeling function L^4. This can again be explained by the domain sizes of the labeling functions, but also the fact that the bounds obtained with L^4 can be expected to be closer to the optimal solution values and, hence, it becomes increasingly more difficult to find better

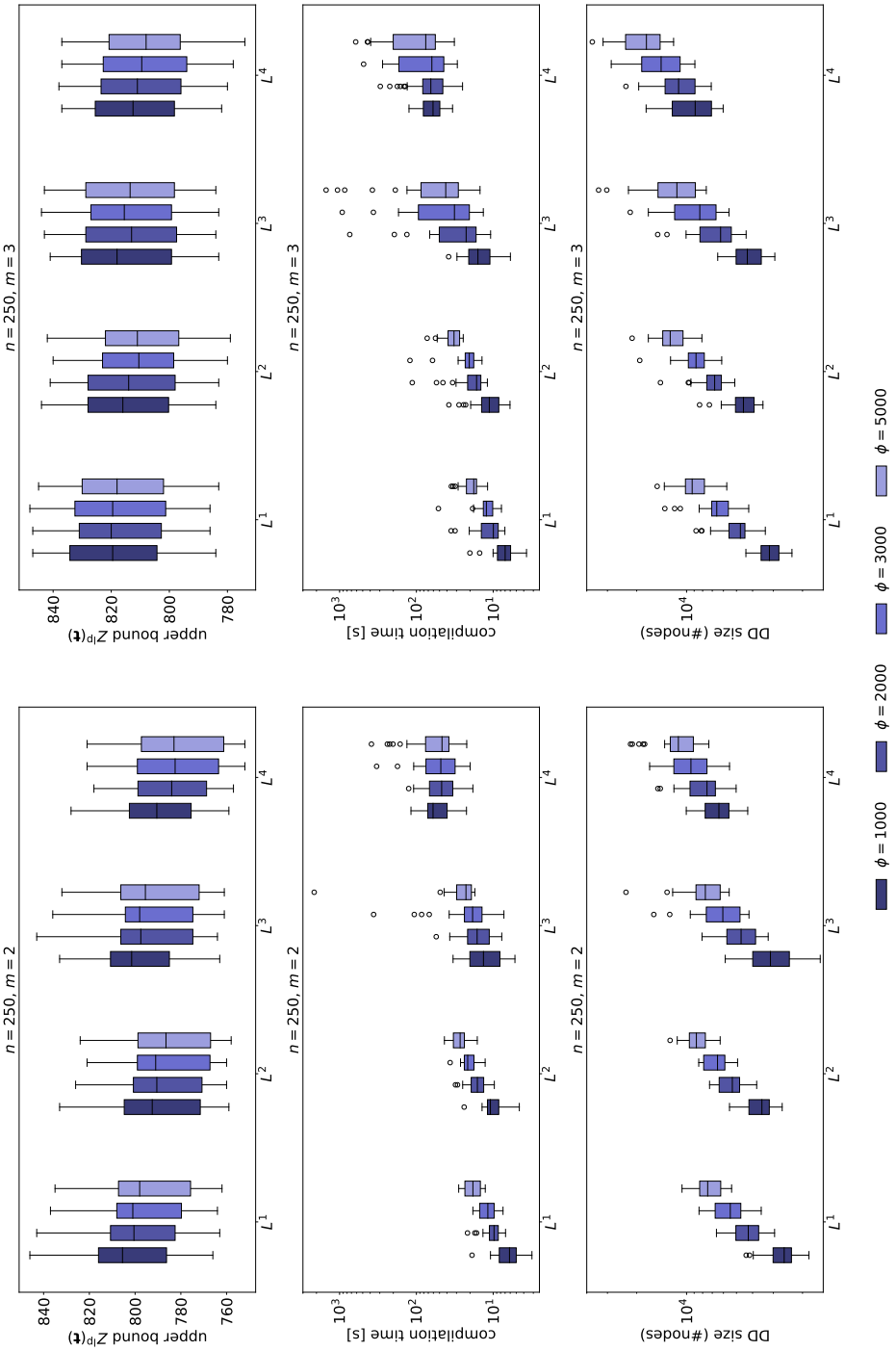


Figure 4.4: A*C results obtained for type P^B instances with 250 jobs and $m \in \{2, 3\}$ secondary resources in dependence of the open list size limit ϕ and the used labeling function L^1, L^2, L^3, or L^4.

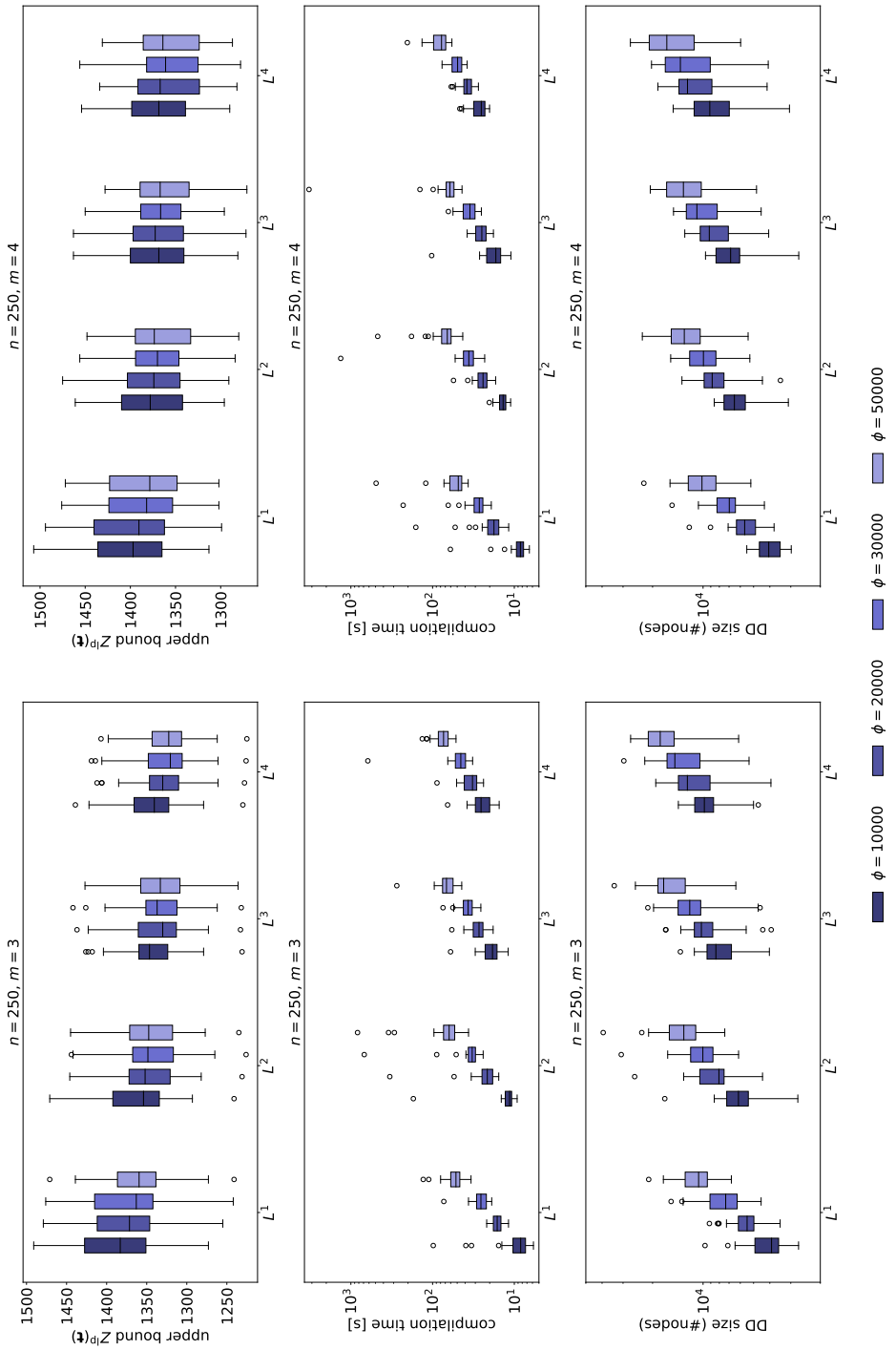


Figure 4.5: A*C results obtained for type A instances with 250 jobs and $m \in \{3, 4\}$ secondary resources in dependence of the open list size limit ϕ and the used labeling function L^1, L^2, L^3, or L^4.

bounds. When comparing the results of L^2 and L^3, we can see that L^2 usually yields marginal better results, but this again at the cost of longer running times.

Figure 4.5 considers instances of type A and shows results for $\phi \in \{10000, 20000, 30000, 50000\}$. Note that we used here larger values for ϕ than in the previous experiments for instance set P^{B}. This is due to fixed time horizon of $T = 1000$ in type A instances which implies that the MDDs' heights are restricted correspondingly. Consequently, larger values for ϕ can be used to utilize roughly comparable computation times. Again, we can see that parameter ϕ allows to control the quality of the obtained relaxed MDDs. With larger values for ϕ the lengths of the longest paths of the obtained relaxed MDDs are in general smaller, while the computation times and MDD sizes increase.

For all further experiments, we made a compromise between expected quality of the relaxed MDD and compilation time. We use the following settings. Instance set P^{B}: labeling function $L^3(u)$ and $\phi = 1000$; instance set A: labeling function $L^4(u)$ and $\phi = 20000$.

4.12.4 Upper Bound Comparison

In this section the upper bounds obtained by A*C are compared with the upper bounds from TDC and IRLP. Regarding A*C we will consider the length of the longest paths in the final MDDs as well as the frequently stronger upper bound $Z^{\mathrm{ub}}_{\mathrm{min}}$, which is obtained at the first time the target node is chosen the for expansion. In addition, we compare with the upper bounds obtained by an MILP approach.

The MILP approach relies on the compact order-based formulation from Horn et al. [56] (see Appendix A.2) and has been solved by the Gurobi Optimizer Version 7.5.1[2] in single-threaded mode with a CPU time limit of 900 seconds. The IRLP compilation is performed with a CPU time limit of 900 seconds and the TDC approach is executed with two different width limits β which are chosen s.t. the average running times are in the same order of magnitude and are usually not smaller than those of A*C: We use $\beta \in \{300, 500\}$ for set P^{B} and $\beta \in \{3000, 5000\}$ for set A.

Figures 4.6 and 4.7 show the results of this comparative study for instance sets P^{B} and A, respectively. The diagrams in the first row show the obtained upper bounds, the second row's diagrams the computation times, and the diagrams in the third row depict the obtained relaxed MDDs' total number of nodes. Each group of bars on the horizontal axes corresponds to a specific instance class with the stated number of jobs. Each bar indicates the average value over all 30 instances of the corresponding instance class and the respective approach.

Concerning the depicted computation times, each first bar shows A*C's average time to obtain $Z^{\mathrm{ub}}_{\mathrm{min}}$, i.e., the time when the construction would be stopped according the classical A* termination criterion. The second bar shows the average time required for the construction of the complete relaxed MDD. Since the MILP approach as well as

[2] http://www.gurobi.com

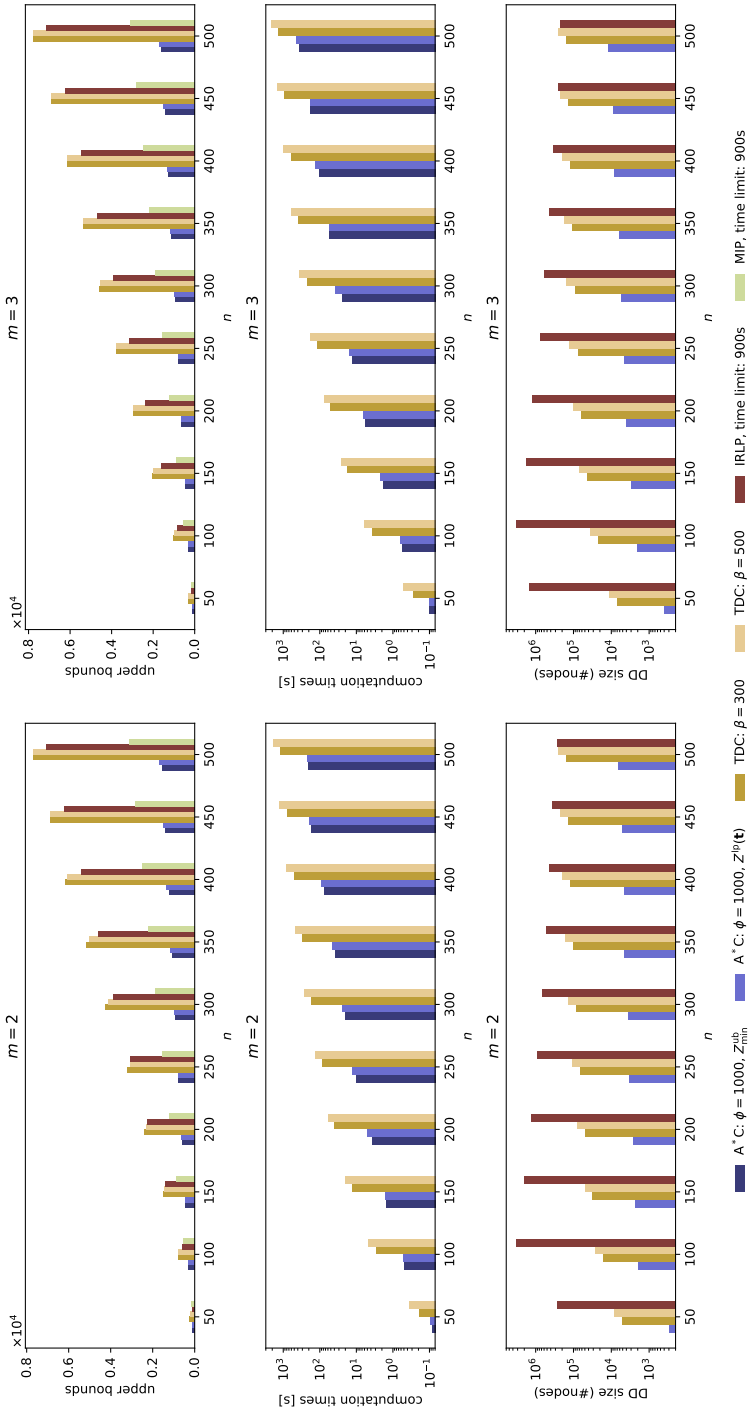


Figure 4.6: Average values for upper bounds, computation times, and MDDs-sizes obtained from A*C, TDC, IRLP, and the order-based MILP approach on instance set P^B.

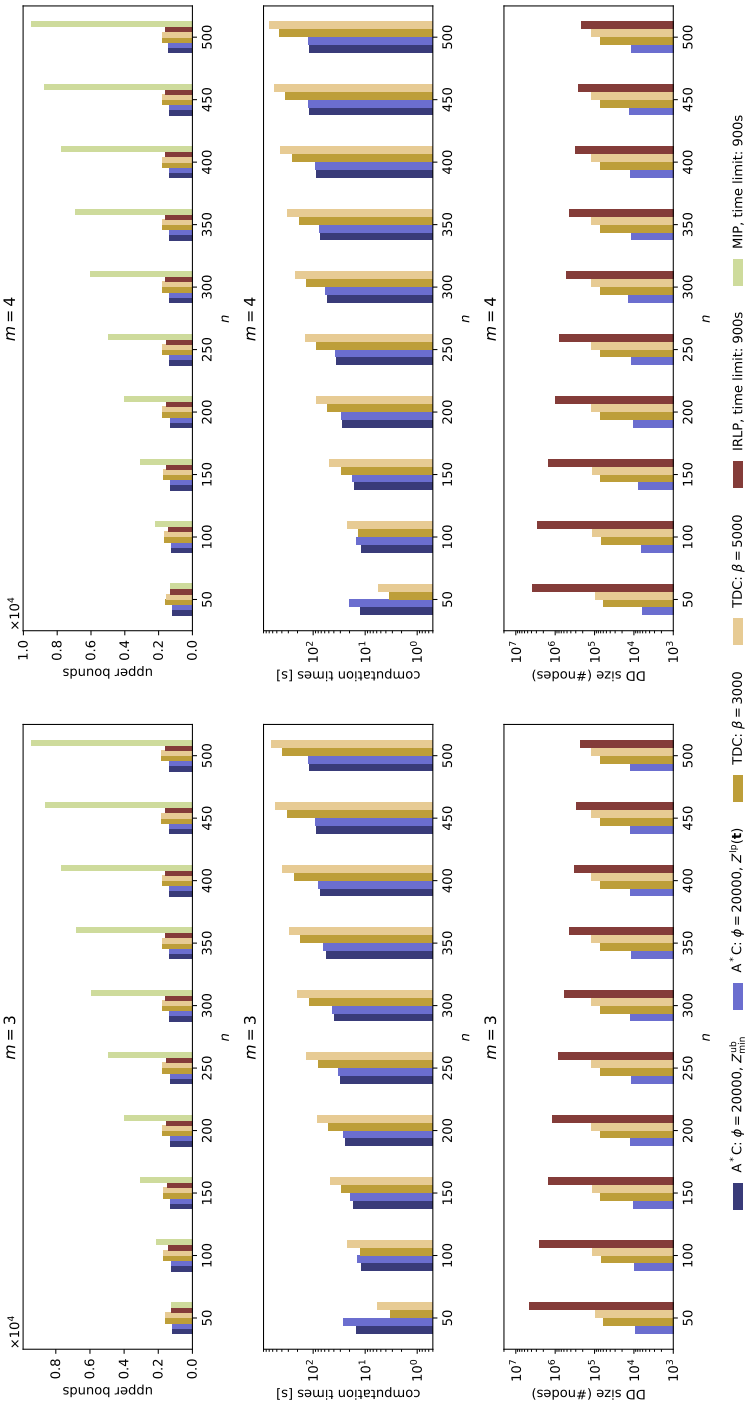


Figure 4.7: Average values for upper bounds, computation times, and MDDs-sizes obtained from A*C, TDC, IRLP, and the order-based MILP approach on instance set A.

IRLP exhausted the time limit of 900 seconds in almost all runs, we omit corresponding bars as they would not give more insight. More specifically, the MILP solver could only prove optimality for the following percentages of the smallest instances with $n = 50$ jobs: 30% and 10% of set P^B for $m \in \{2, 3\}$, respectively, and 3.33% of set A for $m = 3$. For $m = 4$ the MILP solver was not able to solve any instance to proven optimality.

The results for instance set P^B, shown in Figure 4.6, give a rather clear picture. The average upper bounds Z^{ub}_{min} obtained by the A*C algorithm are always the strongest. They are in particular substantially better than those obtained from the TDC variants and the IRLP approach. The difference is more than a factor of four for the largest instances. Even more major are the differences in the sizes of the respectively obtained MDDs. A*C's MDDs are usually more than an order of magnitude smaller than those compiled with TDC and IRLP. The A*C algorithm takes here clear advantage from the feature that a node may be reached from the root node via multiple paths consisting also of different numbers of transitions. This flexibility obviously is further exploited by the careful selection of nodes that are merged. Remember that in contrast, TDC and IRLP are strongly layer-oriented and no merges across layers are possible. Moreover, multiple nodes at different layers are frequently required for the same state. The bounds obtained from the MILP approach are clearly better than those of TDC and IRLP, but also significantly worse than those of A*C. Differences between Z^{ub}_{min} and the length of longest paths in the final MDDs are in comparison to the bounds from the other approaches not that large, but still notable. Thus, completing the construction of the relaxed MDD clearly only pays off if the relaxed MDD is utilized in some further way, like our successive derivation of a restricted MDD from Section 4.10.

For instance set A, Figure 4.7 shows remarkable differences. The upper bounds obtained from the MILP approach are far worse than those obtained from A*C as well as TDC and IRLP. Differences between A*C, TDC, and IR are not that large anymore, nevertheless, in every case the strongest average upper bounds could be obtained by A*C. The better performance of the classical approaches TDC and IRLP on these instances in comparison to set P^B can be explained by the constant time horizon and the special prize structure, due to which the height of the MDDs is limited in a stronger way. Concerning the size of the obtained MDDs, A*C exhibits again substantial advantages over TDC: A*C's MDDs only have about half the size of TDC's MDDs, and those of IRLP are even more than three times larger than those of A*C for the smaller instances.

4.12.5 Impact of Filtering

In the next series of experiments we aim at assessing the effectiveness of the proposed filtering from Section 4.9 on the relaxed MDDs obtained by A*C. We investigate first how much a given relaxed MDD can be reduced. Afterwards, we consider the impact of the filtering on the subsequent construction of a restricted MDD.

Next to a relaxed MDD, a lower bound Z^{lb} is required for our filtering. Such a lower bound can be obtained efficiently during the A*-based compilation of the relaxed MDD by

storing the length of an longest path from $\mathbf{r}$ to an exact node. However, especially for large instances, this lower bound is typically weak, since nodes have to be extensively merged during the compilation. Clearly, a stronger lower bound has the potential that more arcs and nodes can be pruned. For this reason, it makes sense to invest more computation time for obtaining a stronger lower bound in order to enhance the effectiveness of the filtering. A way to obtain a frequently stronger bound for the filtering in reasonable time is a preliminary application of our construction of a restricted MDD according to Algorithm 4.4, with a maximum width that is only a small fraction compared to the maximum width used for the final main construction after the filtering.

Figure 4.8 shows the average percentage of arcs that are removed by filtering over all of our benchmark instances. We consider relaxed MDDs constructed by A*C using labeling function L^3 with $\phi = 1000$ for instance set P^{B} and labeling function L^4 with $\phi = 45000$ for instance set A. Different lower bounds are obtained from (1) the longest path to an exact node in the relaxed MDD and (2) by constructing a preliminary restricted MDD of maximum width $\beta \in \{1, 100, 500\}$ and $\beta \in \{1, 15000, 20000\}$ for instance sets P^{B} and A, respectively. We included the maximum width one as this case resembles a classical greedy construction heuristic. The larger values for the maximum width have been determined in preliminary experiments and are set in such a way that the performance boost due to the stronger bound outweighs the required computation time for providing them.

When using the longest path to an exact node as lower bound, the filtering performs poorly. For benchmark set P^{B}, filtering only has an effect for instances with 50 and 100 jobs. For instance set A, the percentage of removed arcs is for most instance sizes rather low. When lower bounds obtained by constructing a restricted MDD are used, filtering becomes considerably more effective. Already with a maximum allowed width of one, on most instance sizes between 30% and 50% of all arcs can be removed. Increasing the maximum width for the restricted MDD construction allows filtering to a larger extent. Lower bounds obtained with the next higher considered maximum width (i.e., 100 for set P^{B} and 15000 for set A) allows pruning on average 12% and 43% more arcs for instance sets P^{B} and A, respectively. If the maximum allowed width is further increased to 500 and 20000, respectively, on average additional 1.4% and 0.5% of the arcs from the given MDDs are removed. This indicates that increasing the maximum allowed width for preliminary compiled restricted MDDs further will have a decreasingly noticeable effect.

Our main motivation for filtering a relaxed MDD is to provide a stronger starting point for the main compilation of a restricted MDD. To assess the impact of filtering using the different lower bounds we perform the successive compilation of restricted MDDs with fixed maximum widths of 12000 and 450000 for benchmark sets A and P^{B}, respectively. The motivation for these choices is that the whole approach requires on the largest benchmark instances about 900 seconds to complete.

Figure 4.9 shows median total computing times for the heuristic exploitation of the produced relaxed MDDs. These processing times include the times for obtaining a preliminary lower bound by the different strategies, the application of the filtering, and

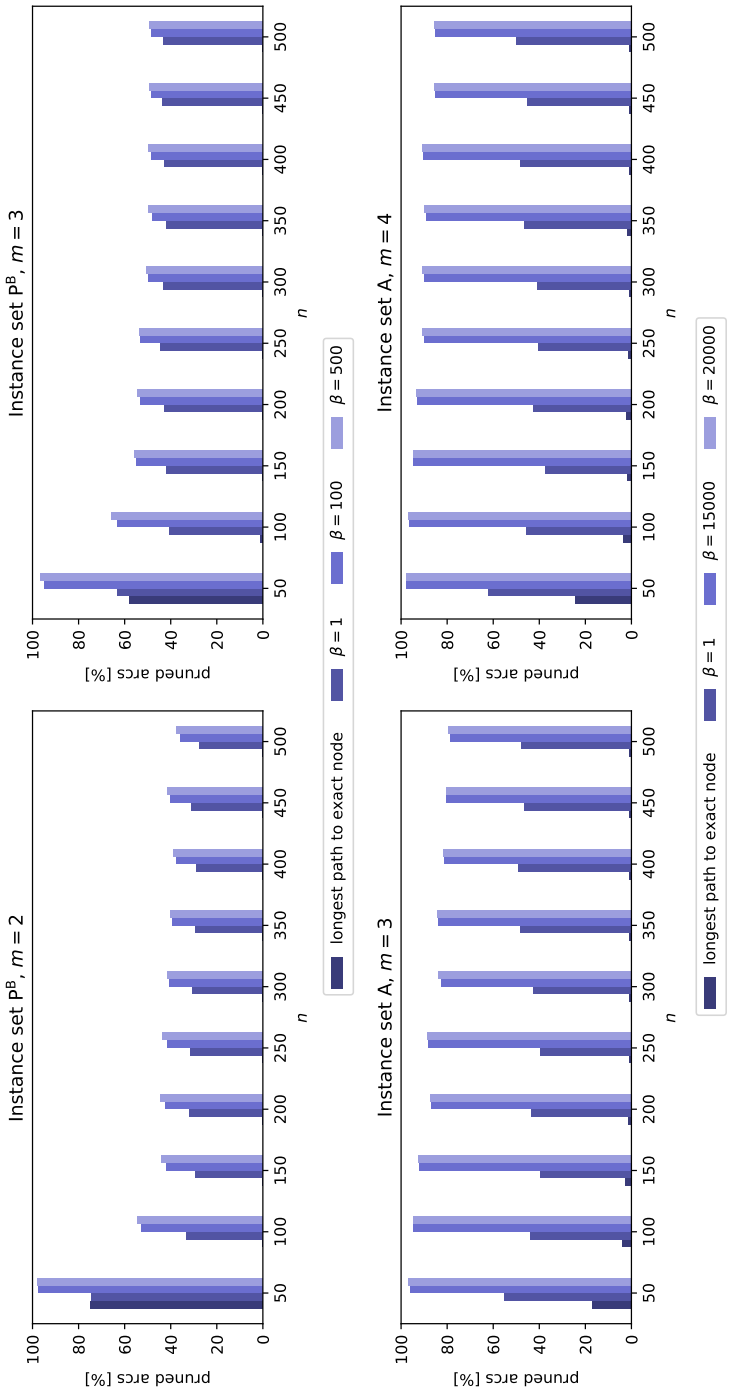


Figure 4.8: Percentage of arcs pruned depending on the used lower bound. The lower bounds are either obtained from the longest path to some exact node or by restricted MDDs of maximum width β.

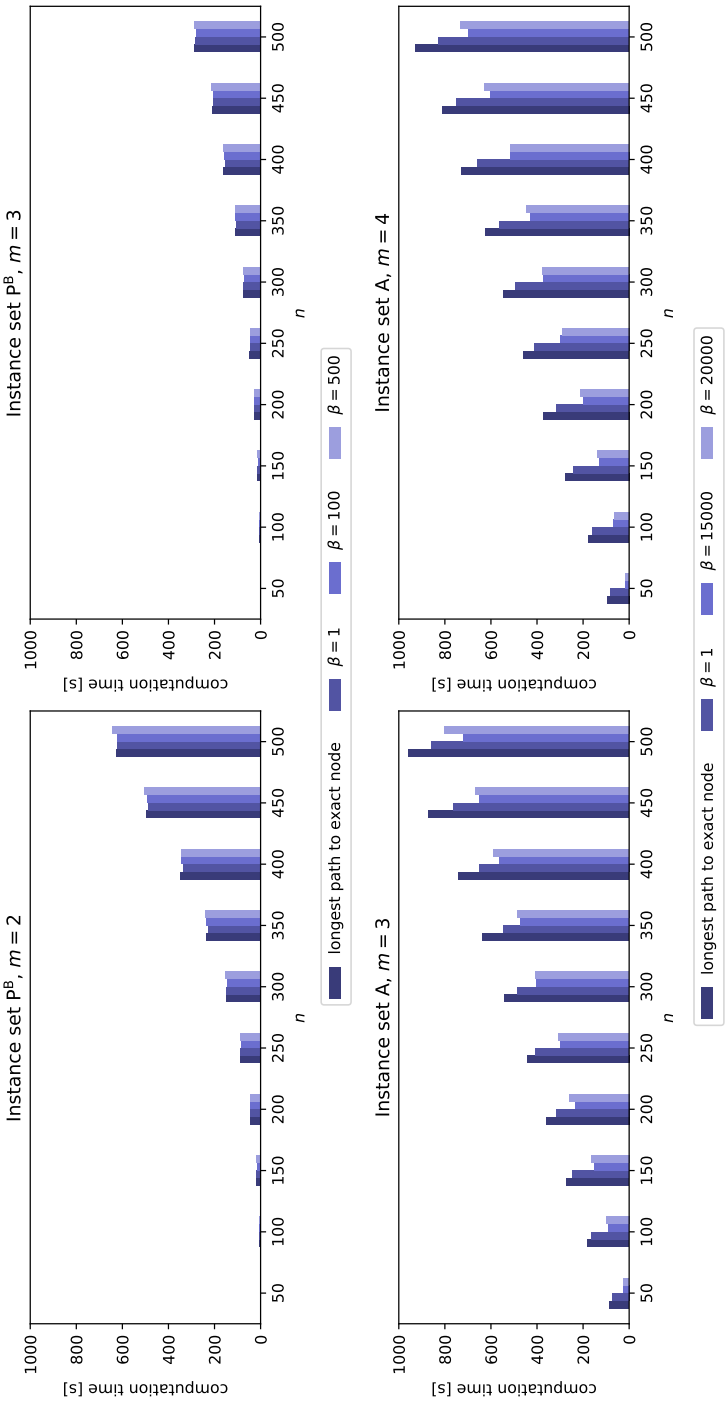


Figure 4.9: Median total computation times for obtaining different primal bounds used for filtering and the subsequent compilation of restricted MDDs.

the compilation of the final restricted MDD. Although the differences among the times for the different lower bound methods during the filtering are small on instance set P^B, there is a clear trend observable. If the lower bound obtained from the longest path to an exact node is used, we get in almost all cases the longest computation times. Since these bounds are too weak to allow a significant pruning of arcs, especially on medium to large instances, the shown times essentially match the times if we would not apply filtering at all. If we obtain a lower bound for the filtering from a small restricted MDD, then the computation times are typically shorter. However, with a larger maximum width β also the computational effort for providing the lower bounds increases, which at some point outweighs the advantage of a better filtered relaxed MDD. For benchmark set P^B, this is frequently the case with $\beta = 500$. The performance improvements due to the filtering of the relaxed MDDs are more pronounced on the instances of type A. Again, there is already a small performance improvement observable if we use the lower bound from restricted MDDs of width one. In contrast to benchmark set P^B, there is a later break-even point for when the time spent to obtain a better lower bound for the filtering outweighs the performance gains. This allows to use a larger maximum width in order to likely obtain a stronger lower bound, which in turn enables to prune more arcs, and ultimately perform the final compilation of the restricted MDD in considerably shorter time.

From the processing times shown in Figure 4.9, only a small part is required for the filtering procedure itself. For instance set P^B and A up to 3% and 6%, respectively, of the depicted computation times are used for the filtering. The quality of the final solutions obtained from the main restricted MDD have shown to be largely independent of the filtering. We interpret this as a sign for the robustness of the chosen greedy criterion for selecting nodes to remove during the construction of the restricted MDD and not as a weakness of the filtering procedure.

4.12.6 Lower Bound Comparison

We finally want to compare the A*C approach to construct a relaxed MDD plus the followup construction of a restricted MDD to obtain a heuristic solution to other solution methods for larger PC-JSOCMSR instances. Now, our focus is primarily on the quality of obtained heuristic solutions, i.e., on lower bounds, but clearly our approach also yields upper bounds from the relaxed MDD and we will study resulting gaps. We compare to TDC for restricted MDDs, the GVNS metaheuristic, an MILP approach, and a basic CP formulation.

As in Section 4.12.5 we apply A*C using the labeling function L^3 with $\phi = 1000$ and labeling function L^4 with $\phi = 20000$ for the instance sets P^B and A, respectively. Preliminary lower bounds for the filtering are obtained by compiling restricted MDDs of maximum width 100 for type P^B instances and 15000 for type A instances. Afterwards, filtering is performed. Finally, the main restricted MDDs are compiled with $\beta = 12000$ and $\beta = 450000$ for benchmark sets P^B and A, respectively.

TDC's maximum allowed width is set to 2000 and 12000 for instance sets P^B and A, respectively. These values have been selected in such a way that the TDC terminates for the largest instances in about 900 CPU-seconds. For the GVNS we use again the same configuration as described in Section 4.11 and compare to the best found solution after 900 seconds. Moreover, we compare to the objective values of the best feasible solutions provided by the order-based MILP formulation from Horn et al. [56], see Appendix A.2, solved again by Gurobi Optimizer 7.5.1[3] using a single thread with a CPU time limit of 900 seconds. Last but not least, we also consider the CP model from [56], see Appendix A.3. The model was implemented with MiniZinc 2.1.7[4] and we apply the backbone solver Chuffed with a time limit of 900 seconds.

The results of all approaches are presented in Table 4.5. Each row shows the aggregated results over the 30 benchmark instances with the characteristics given in the first three columns. For all approaches, columns labeled $\overline{obj}$ and $\sigma(obj)$ state the mean objective values of obtained heuristic solutions and corresponding standard deviations. For the MDD-based approaches these values correspond to the lengths of the longest paths in the restricted MDDs. Moreover, we list for the MDD-based approaches median total computation times in seconds in the $t[s]$ columns. For A*C+TDC median computation times are also given for its individual steps. Column $t^f[s]$ states median times just for filtering the relaxed MDDs including the times for determining the required lower bound. Furthermore, in column $t^c[s]$ median times for compiling the final restricted MDDs are shown. For GVNS, MILP, and CP timing information is omitted as they were always terminated with the time limit of 900 seconds. The only exceptions are MILP and CP runs for the smallest instances with 50 jobs, which finished in some cases earlier with proven optimality. In addition, we list for our A*C+TDC average optimality gaps in $\overline{\%\text{-gap}}$ which are calculated by $100\% \cdot (Z^{ub}_{min} - obj)/Z^{ub}_{min}$.

If we disregard the results from the benchmark instances of type P^B with three secondary resources for now, Table 4.5 shows a clear picture. A*C+TDC provides in general the best solutions, followed by the GVNS and the TDC. While the TDC performs on the P^B instances with two secondary resources in most cases better than the GVNS, the GVNS is superior to the TDC on the other instances. The weakest solutions have on average been obtained by the MILP and CP approaches, which are only competitive for type P^B instances with 50 jobs. Especially, for the medium to large instances theA*C+TDC typically requires less time than TDC. The superiority of our method compared to the conventional TDC in two ways. Not only are we able to construct much larger restricted MDDs, usually yielding better solutions in less time, but since we first also determine the relaxed MDDs, our approach has the additional bonus of providing upper bounds. Average gaps never exceed 15% and are in particular for instance set A usually not larger than 8%.

On benchmark instances of set P^B with three secondary resources, GVNS typically provides the best solutions when more than 150 jobs are considered. The relative

[3] http://www.gurobi.com
[4] https://www.minizinc.org

			A*C+TDC					TDC			GVNS		MILP		CP	
type	m	n	$\overline{obj}$	$\sigma(obj)$	%-gap	$t^f[s]$	$t^c[s]$	$t[s]$	$\overline{obj}$	$\sigma(obj)$	$t[s]$	$\overline{obj}$	$\sigma(obj)$	$\overline{obj}$	$\sigma(obj)$	$\overline{obj}$	$\sigma(obj)$
P^B	2	50	**123.3**	10.3	2.2	<1	<1	<1	**123.3**	10.3	<1	123.2	10.4	122.8	10.5	**123.3**	10.3
P^B	2	100	**259.9**	11.7	9.4	<1	5	6	259.2	11.7	5	259.0	11.9	240.0	12.2	200.7	20.0
P^B	2	150	**401.2**	18.9	11.5	<1	17	19	398.8	19.3	20	396.6	17.7	331.2	21.2	261.0	23.5
P^B	2	200	**530.0**	18.7	12.3	<1	43	50	526.1	19.3	54	527.1	19.0	424.0	25.3	273.6	61.2
P^B	2	250	**667.2**	21.1	12.6	1	84	100	661.7	20.0	108	660.8	22.4	523.0	30.8	281.0	52.9
P^B	2	300	**797.4**	16.8	13.1	2	145	170	792.0	16.8	193	790.0	17.5	607.4	32.1	308.1	78.5
P^B	2	350	**931.3**	25.6	13.7	3	231	282	923.9	25.2	313	923.0	26.7	685.6	34.4	326.7	122.2
P^B	2	400	**1061.6**	21.3	13.9	5	338	442	1054.4	21.4	481	1055.1	19.4	741.9	47.7	329.6	110.9
P^B	2	450	**1197.0**	28.7	13.5	7	485	679	1187.5	28.6	704	1180.9	27.4	807.6	55.7	348.6	152.4
P^B	2	500	**1339.1**	25.4	14.6	8	613	845	1330.3	25.0	949	1324.3	28.0	864.9	74.7	403.4	351.5
P^B	3	50	**140.3**	10.4	1.3	<1	<1	<1	**140.3**	10.4	<1	**140.3**	10.4	139.0	10.5	140.2	10.5
P^B	3	100	**289.9**	14.5	5.1	<1	5	6	288.4	14.6	7	288.5	14.3	270.4	17.0	240.0	16.4
P^B	3	150	**437.4**	15.1	7.1	<1	13	15	433.5	14.8	24	437.0	17.1	378.9	20.0	331.3	19.3
P^B	3	200	581.8	18.3	8.4	<1	27	33	576.9	18.2	59	**582.9**	16.3	495.8	18.8	367.2	44.4
P^B	3	250	716.7	13.6	9.5	1	44	63	712.0	14.4	117	**721.9**	16.5	603.6	22.4	380.6	89.3
P^B	3	300	850.3	16.0	11.2	2	71	107	846.0	15.3	210	**864.0**	19.8	702.7	27.4	419.5	109.8
P^B	3	350	988.0	26.8	11.7	3	107	171	983.3	27.7	341	**1008.2**	29.3	782.9	35.4	405.7	188.8
P^B	3	400	1124.6	24.5	12.3	5	152	296	1119.1	23.6	530	**1142.5**	24.1	877.8	45.2	527.5	161.1
P^B	3	450	1266.3	19.7	11.9	7	198	433	1257.7	20.9	751	**1283.5**	26.1	951.6	53.6	524.5	207.3
P^B	3	500	1397.8	25.3	12.3	10	269	672	1392.1	23.7	1082	**1418.0**	27.1	1035.1	60.6	589.3	238.6
A	3	50	**1130.3**	39.8	5.0	7	20	57	1127.8	38.4	6	**1130.3**	39.8	1086.8	46.9	892.0	45.8
A	3	100	1201.1	36.5	5.6	9	81	110	1196.6	36.4	23	**1201.4**	37.0	1084.8	60.5	712.7	44.1
A	3	150	**1215.5**	26.3	6.0	11	141	169	1208.9	28.7	58	**1215.5**	27.0	972.4	58.4	643.4	41.9
A	3	200	1228.9	21.8	6.7	14	219	261	1220.4	25.6	109	**1229.4**	21.4	900.4	67.1	544.5	149.4
A	3	250	1244.7	28.6	6.6	18	279	331	1238.5	30.8	180	**1245.4**	28.7	900.0	73.3	575.3	48.0
A	3	300	**1243.9**	23.4	7.5	22	379	448	1234.4	22.9	265	1243.7	23.5	736.8	109.7	553.8	41.3
A	3	350	**1256.2**	22.6	7.4	30	440	542	1245.6	24.8	370	1255.5	23.6	677.9	78.5	536.5	50.3
A	3	400	**1269.7**	19.1	8.0	34	529	646	1262.5	19.1	493	1267.2	19.6	668.0	70.1	525.2	42.8
A	3	450	**1268.4**	19.2	8.3	41	609	748	1257.5	24.5	647	1268.2	18.3	694.9	79.5	516.6	48.6
A	3	500	**1271.7**	19.0	8.2	46	676	869	1260.4	22.7	799	1271.2	17.9	669.4	61.6	527.5	24.0
A	4	50	1141.8	35.5	3.4	7	2	45	1138.7	35.0	6	**1142.4**	36.2	1105.5	41.2	882.5	42.0
A	4	100	1218.8	40.6	4.0	8	57	82	1215.7	41.5	25	**1218.9**	40.6	1102.5	64.7	708.9	119.8
A	4	150	**1253.8**	30.6	4.4	10	118	146	1248.5	30.4	60	1253.5	30.9	997.3	55.5	655.6	48.5
A	4	200	1259.5	31.3	4.9	15	180	228	1253.6	32.0	118	**1259.6**	31.7	913.5	65.2	576.5	129.5
A	4	250	**1280.4**	27.0	5.9	20	277	339	1273.6	25.7	191	**1280.4**	28.2	937.0	85.4	541.4	134.8
A	4	300	**1293.8**	25.1	5.9	27	346	441	1282.6	28.0	287	1292.3	24.9	759.1	126.6	565.0	46.0
A	4	350	**1298.9**	23.9	5.8	32	394	504	1289.9	24.5	396	1297.2	22.9	701.8	73.2	548.8	38.1
A	4	400	**1304.4**	42.0	6.0	37	477	617	1298.1	41.2	533	1299.6	42.9	700.6	72.5	541.4	27.9
A	4	450	**1308.4**	25.2	6.9	45	551	752	1298.9	29.2	672	1304.7	26.7	702.7	64.0	546.5	56.6
A	4	500	**1315.6**	28.0	6.9	48	644	842	1304.0	25.8	858	1312.5	29.1	691.9	54.3	526.8	33.1

Table 4.5: Comparison of the subsequent application of A*C, filtering, and the construction of restricted MDDs to the TDC for restricted MDDs, the GVNS, the MILP, and the CP approach.

differences between the obtained objective values from GVNS and our A*C-based approach are typically about one to two percent. We believe that in these cases, the GVNS's local search is particularly effective. Clearly, an option would be to try to finally "polish" the MDD-based methods' solutions by applying a local search. Another particularity of the results for this instance set are the required times $t^{c}[s]$ for constructing the main restricted MDD based on the relaxed MDD. Although the same maximum width is used as for the instances with two secondary resources, these median times are considerably shorter for the case with three secondary resources. This indicates that the removal criterion rates many nodes the same. However, it also showcases the performance improvements of the compilation technique when the information contained in the relaxed MDD is utilized.

The optimality gaps increase with the problem size on both instance sets, as one might expect for a compilation of relaxed and restricted MDDs with fixed parameter values. In comparison to instance set A, we obtain smaller optimality gaps on type P^{B} instances with few jobs but get larger optimality gaps for the instances with many jobs. This can be explained by the problem size independent time horizon of set A instances, which implies a certain maximal number of jobs that can be scheduled independently of the number of available jobs. Consequently, the relative difference between the lower and upper bound is bounded to some degree by this maximal number of jobs. Moreover, for some instances the optimality gap has been closed, i.e., they could be solved to proven optimality. This was the case for nine of the type P^{B} instances with two secondary resources and 50 jobs. For type P^{B} instances with three secondary resources we could optimally solve ten instances that consider 50 jobs. Furthermore, for a single benchmark instance with 50 jobs and four secondary resources of type A, the lower and upper bound coincided.

4.13 Conclusions

In this chapter we studied the application of multivalued decision diagrams (MDDs) for the prize-collecting job sequencing with one common and multiple secondary resources (PC-JSOCMSR) problem. PC-JSOCMSR is strongly related with scheduling treatments within individual days in the context of the particle therapy treatment center considered in Chapter 3. However, unlike PTPSP the here studied PC-JSOCMSR is a much more general problem which is demonstrated by showing that PC-JSOCMSR also appears as a sub-structure in the pre-runtime scheduling of electronics within an aircraft.

We started by providing a recursive model for PC-JSOCMSR and showed how MDDs can be obtained from the problem's state graph. Whenever, the size of MDDs become to large relaxed and restricted MDDs are employed to obtain dual bounds and heuristic solutions, respectively. We adapted the two main compilation techniques for relaxed MDDs proposed in the literature to PC-JSOCMSR: top-down construction (TDC) and incremental refinement (IRLP). To obtain restricted MDDs we use a variant of the TDC.

Furthermore, we present a novel way of compiling relaxed MDDs that applies the principles of A* search. To this end, an upper bound on the remaining partial solution

plays an important role to estimate if a node is likely to be part of a longest path or not. Nodes that are not considered promising are merged with its corresponding collector node. Such a collector node is labeled in a state-space-relaxation fashion and nodes can only be merged if they have the same label. Choosing a proper labeling is not only important to obtain a strong relaxation but also to prevent cycling during the construction of the relaxed MDD. The use of collector nodes with a label to define similar states deviates from classical construction principles, which are all strongly layer-oriented. This extension is especially relevant for problems where the cardinality of the solution is not known. Nevertheless, layer-oriented merging can easily be achieved within our framework by including the layer as a part of the label. The constructed relaxed MDD provides an upper bound on the optimal solution value of PC-JSOCMSR, but it also contains important information about promising feasible solutions of the problem. For this reason, we propose constructing a restricted MDD based on the relaxed one, instead of creating a restricted MDD independently, which in the literature so far is the common way of doing it.

For our computational study, we generated in total three benchmark instance sets with different characteristics adopted from the real-world application scenarios. We first compared the relaxed MDDs obtained by TDC and IRLP. While both methods perform rather similar on balanced instances, IRLP is clearly superior on the benchmark instances with skewed workload. We assessed next the quality of the restricted MDDs compiled with the TDC by comparing the obtained heuristic solutions with the ones from an independent general variable neighborhood search (GVNS). While the TDC performs better than the GVNS on small to medium-sized instances, the GVNS is mostly superior on the largest instances of our benchmark sets.

Afterwards, we considered the A*-based approach. We observed that the size of the obtained relaxed MDDs can be controlled by the open list size limit ϕ as well as the used labeling function. The constructed relaxed MDDs are frequently an order of magnitude smaller than those compiled with the TDC and provide upper bounds that are usually better by a substantial factor. The successive compilation of a restricted MDD based on a relaxed MDD in order to obtain heuristic solutions and lower bounds also turned out to be effective. The main benefit is a substantial speedup in the construction of the restricted MDD. We even showed that the total time for constructing the relaxed MDD, filtering it, and deriving a restricted MDD of a certain size based on the relaxed MDD can take less time than the classical independent construction of a restricted MDD of the same size.

We further compared our MDD-based methods to an order-based MILP model solved by Gurobi, to the GVNS, and to a CP approach solved by MiniZinc with Chuffed as backend. For most cases, our A*-based approach of deriving a relaxed MDD followed by the construction of a restricted MDD yielded the best solutions. An exception are the larger instances of the particle therapy benchmark set with three secondary resources, where the GVNS clearly outperformed the other approaches. The MILP and the CP model produced rather weak lower bounds for all instances except the smallest.

The idea to compile relaxed MDDs with principles of A*-search that avoids an explicit consideration of layers originates from the fact that the cardinality of solutions for a PC-JSOCMSR instance varies. In future work it would be of interest to investigate if the A*-based approach is also effective for problems where the cardinality solutions is fixed and the layer is included in the label. Another topic to investigate in more detail is how the choice of the open list size limit ϕ and the labeling function relates to the actual size of the constructed MDDs. The selection of suitable values for ϕ is unfortunately not that obvious and so far we relied on preliminary tests. Although our approach could solve several instances to proven optimality, it is not primarily designed to converge to proven optimal solutions. A further step would be to use the new MDD compilation methods within algorithmic frameworks that iteratively try to get closer to a proven optimal solution. For example highly promising are novel branching schemes on the basis of relaxed MDDs as described in Bergman et al. [13] or the utilization of MDDs in new inference techniques in constraint programming, see e.g. Cire and Hoeve [23]. Last but not least, it should be promising to explore also other possibilities for deriving restricted MDDs from relaxed ones.

CHAPTER 5

Strip Packing and Resource-Constrained Project Scheduling

This chapter is devoted to two works that both propose a decomposition approach. The first considers a strip packing problem and provides a logic-based Benders decomposition. The approach has been presented at the *International Conference on Operations Research* (OR'15) and has been published in the proceedings of the conference:

> J. Maschler and G. R. Raidl. A logic-based Benders decomposition approach for the 3-staged strip packing problem. In *Operations Research Proceedings 2015*, pages 393–399. Springer, 2017.

As this work predates our project with MedAustron it is not directly related to patient scheduling in particle therapy. Nevertheless, strip packing has many relationships to the scheduling problems considered in this thesis from a structural point of view. In strip packing rectangles have to be placed onto a strip of fixed width and unlimited height. There is a correspondence between the strip packing's rectangles and the jobs of the scheduling problems: While rectangles have a width and a height, jobs have resource requirements and a processing time. Moreover, the strip's width translates to some degree to the number of available resources in our scheduling problems.

The second presented work in this chapter proposes a time-bucket relaxation for a resource-constrained project scheduling variant. The considered variant is in some way the predecessor of PC-JSOCMSR in that it addresses the PTPSP's time assignment of a single day. In contrast to PC-JSOCMSR, however, we adapt here the original definition of DTs in which they have been composed of activities. The suggested approach starts

with a relaxation in which time is aggregated into so-called buckets which are then iteratively refined. Next to an extended abstract for the *6th International Workshop on Model-based Metaheuristics* (Matheuristics 2016) there is a co-authored publication accepted in *International Transactions in Operational Research*:

> G. R. Raidl, T. Jatschka, M. Riedler, and J. Maschler. Time-bucket relaxation based mixed integer programming models for scheduling problems: A promising starting point for matheuristics. In *Proceedings of Matheuristics 2016: 6th International Workshop on Model-Based Metaheuristics*, pages 104–107, Brussels, Belgium, 2016,
>
> 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*, 2017.

Moreover, a master's thesis [61] has been completed on this subject. We give here only an overview of the method's most central aspects and the key results, as the contribution of the author of this thesis was considerably smaller than the ones of the other authors. To this end, Section 5.2 is a revised and updated version of the extended abstract. For details, we refer to [104, 61, 103].

5.1 3-Staged Strip Packing

We consider the *3-staged strip packing problem* (3SPP), in which rectangular items have to be arranged onto a rectangular strip of fixed width, such that the items can be obtained by three stages of guillotine cuts while the required strip height is to be minimized. We propose a new *logic-based Benders decomposition* (LBBD) with two kinds of Benders cuts and compare it with a compact ILP formulation.

5.1.1 Introduction

In the *3-staged strip packing problem* (3SPP) we are given n rectangular items and a rectangular strip of width W_S and unlimited height. The aim is to pack the items into the strip using minimal height, s.t. all items can be received by at most three stages of guillotine cuts. In the first stage the strip is cut horizontally from one border to the opposite one and yields up to n levels. In the second stage the levels are cut vertically and at most n stacks are received. In the third stage the stacks are cut again horizontally and the resulting rectangles of the three consecutive stages of guillotine cuts correspond to the n items and the waste.

The general *strip packing problem* (SPP) was proposed by Baker et al. [4] and has received a large amount of attention: on the one hand many real-world applications, such as glass, paper and steel cutting, can be modeled as SPPs; on the other hand it is strongly-NP hard and has turned out to be a demanding combinatorial problem. We study the special case of the SPP, where only guillotine cuts are allowed as already considered by Hifi [50]

and Lodi et al. [74]. We restrict ourselves to three stages of guillotine cuts. This can be motivated from glass cutting [98].

One of the leading exact approaches for the SPP proposed by Côté et al. [29] is based on a Benders decomposition using combinatorial Benders cuts, which can be seen as an implementation of the LBBD introduced by Hooker and Ottosson [52]. Their master problem cuts items into unit-width slices and solves a parallel processor problem that requires all slices belonging to the same item have to be packed next to each other. The subproblem consists of transforming a solution of the master problem into a solution of the SPP. However, this algorithm cannot be trivially extended to solve the 3SPP.

In this work we suggest a different form of LBBD specifically for the 3SPP, compare it to a compact ILP formulation and show that its performance is competitive. The proposed master problem assigns items to levels and defines the number of stacks in which the items can appear. The resulting subproblems pack items of the same width assigned to the same level into the given number of stacks. Two kinds of Benders cuts are provided, from which the first one are rather straightforward, while the second one are more general.

5.1.2 A Logic-based Benders Decomposition for 3SPP

The proposed LBBD consists of a master problem which is a relaxation of the 3SPP. From an optimal solution of the master problem subproblems are derived that yield a complete solution for the 3SPP. If this solution is not yet optimal we improve the master problem with Benders optimality cuts and resolve the master problem.

Master Problem

The master problem considers n levels and aims to pack the items into these levels. For symmetry breaking item i is only allowed to be packed into level j if $i \geq j$ and only if item j is also packed into level j. This way of symmetry breaking has been already used by Puchinger and Raidl [98]. To better exploit scenarios where many items have the same widths and/or lengths, let us more precisely define the set of all appearing widths as $W = \{w_1, \ldots, w_p\}$, the set of all appearing heights as $H = \{h_1, \ldots, h_q\}$, and the dimensions of item $i \in I = \{1, \ldots, n\}$ as w_{ω_i} and h_{λ_i} with $\omega \in \{1, \ldots, p\}^n$ and $\lambda \in \{1, \ldots, q\}^n$. We further assume w.l.o.g. that the items and the widths in W are given in a non-decreasing order.

We consider in the master problem different variants of each item, in which depending on a parameter $e \in \{1, 2, \ldots\}$ its width is increased by $w_{\omega_i} * e$ while its height is decreased by h_{λ_i}/e. We denote the modified width of an item i by the parameter e with $w_{\omega_i e}$, and analogously, the modified height with $h_{\lambda_i e}$. We require that for each level all packed items of the same width have to be reshaped by the same factor e. The variant of the items models on how many stacks the items are meant to be placed. The total height of the packed variant of items with the same width represents the height if the items

can be partitioned between the stacks ideally. Therefore, the master problem is indeed a relaxation of the original problem since it models the first two stages but not the third.

The next consideration concerns the maximal number of variants that has to be provided for each item. For a given item, the parameter e can be restricted on the one hand by the width of the strip and on the other hand by the total number of items of that width that can be packed into the considered level. The maximal number of different variants for a level j and a width w_g is

$$e_{\max}(j,g) = \min\left(\left\lfloor \frac{W_{\mathrm{S}} - w_{\omega_j}}{w_g} \right\rfloor + \begin{cases} 1 & \omega_j = g \\ 0 & \text{otherwise} \end{cases}, \; |\{i = j,\ldots,n \,|\, \omega_i = g\}|\right).$$

The first term of the minimum calculates the maximum number of items of width w_g that can be placed next to each other without exceeding W_{S}. The second term yields the number of items of width w_g that can be packed into level j.

The master problem is modeled by using three sets of variables: Binary variables x_{jie} which are 1 if item i in variant e is assigned to level j and 0 otherwise. Binary variables y_{jge} which are set to 1 iff an item in variant e with original width w_g is assigned to level j. Integral variables z_j which are set to the height of the corresponding level j. To ease the upcoming notation we denote with $[i,n]$ the set $\{i,\ldots,n\}$ and with $E(j,g)$ the set $\{1,\ldots,e_{\max}(j,g)\}$. The master problem is defined as follows:

$$\min \sum_{j\in[1,n]} z_j \tag{5.1}$$

$$\text{s.t.} \sum_{j\in[1,i]} \sum_{e\in E(j,\omega_i)} x_{jie} = 1 \qquad \forall i \in [1,n], \tag{5.2}$$

$$x_{jie} \le y_{j\omega_i e} \qquad \forall j \in [1,n],\ \forall i \in [j,n],\ \forall e \in E(j,\omega_i), \tag{5.3}$$

$$\sum_{e\in E(j,g)} y_{jge} \le 1 \qquad \forall j \in [1,n],\ \forall g \in [\omega_j,p], \tag{5.4}$$

$$\sum_{i\in[j,n]\,|\,\omega_i=g} x_{jie} \ge e\, y_{jge} \qquad \forall j \in [1,n],\ \forall g \in [\omega_j,p],\ \forall e \in E(j,g), \tag{5.5}$$

$$\sum_{e_1\in E(j,\omega_j)} x_{jje_1} \ge x_{jie_2} \qquad \forall j \in [1,n-1],\ \forall i \in [j+1,n],\ \forall e_2 \in E(j,\omega_i), \tag{5.6}$$

$$\sum_{g\in[\omega_j,p]} \sum_{e\in E(j,g)} w_{ge}\, y_{jge} \le W_{\mathrm{S}} \qquad \forall j \in [1,n], \tag{5.7}$$

$$h_{\lambda_i}\, x_{jie} \le z_j \qquad \forall j \in [1,n],\ \forall i \in [j,n],\ \forall e \in E(j,\omega_i), \tag{5.8}$$

$$\sum_{i\in[j,n]\,|\,\omega_i=g} h_{\lambda_{ie}}\, x_{jie} \le z_j \qquad \forall j \in [1,n],\ \forall g \in [\omega_j,p],\ \forall e \in E(j,g). \tag{5.9}$$

Inequalities (5.2) force that each item has to be packed in a single variant exactly once. Equations (5.3) link the variables x_{jie} and $y_{j\omega_i e}$. The restriction that items of the same width have to be packed in the same variant is guaranteed by (5.4). Inequalities (5.5)

ensure that items are not meant to be placed on more stacks than the number of items. Constraints (5.6) impose that items can only be packed into a level j iff also item j is packed into level j. Constraints (5.7) disallows that the total width of the packed item variants exceed the strip width W_{S}. Constraints (5.8) and (5.9) ensure that the level is at least as high as every single item in it and at least as high as the total height of all packed items of the same width in their corresponding variant.

Subproblem

The master problem is a relaxation of the 3SPP, since it assumes that items of the same width can be partitioned, s.t. the resulting stacks are of equal height. The subproblems determine the actual packing of the items and with it the exact level height. Thus, the resulting subproblems consist of assigning items, of the same width packed by the master into the same level, into the number of stacks determined by their item variant. The objective is to minimize the height of the highest stack. This problem corresponds to the $P||C_{\max}$ problem [42]. We use for the subproblem a straightforward ILP formulation which was solved in less than a second in all considered instances.

Benders Cuts

The aim of Benders cuts is to incorporate the knowledge obtained in the subproblems back into the master problem. In the simple case a Benders cut states that if a set of items is packed in a certain variant, then the height of the level is at least as high as the result of the corresponding subproblem.

Let $\bar{I}$ and $\bar{e}$ be the set of items and their variant that have defined a subproblem and let $\bar{z}$ be the objective value of a corresponding optimal solution. Moreover, the set J' contains those levels that allow an assignment of items, s.t. the Benders cut can get activated. The simple version of our Benders cuts is

$$\left(\sum_{i\in\bar{I}} x_{ji\bar{e}} - |\bar{I}| + 1\right)\bar{z} \leq z_j \qquad \forall j \in J'. \tag{5.10}$$

These Benders cuts have the disadvantage that they do not affect item assignments differing from I' only in that items are exchanged by congruent items, i.e., items having the same width and height. The extended Benders cuts aim to overcome this drawback. Let $\bar{H} \subseteq H$ be the set of heights of the items from a subproblem defined by $\bar{I}$ and $\bar{e}$. We introduce for each height $h \in \bar{H}$ a binary variable u_h which is set to true iff at least as many items of the same width, height and variant are packed into a level as it has been packed in the considered subproblem. The set $I' \subseteq I$ contains all items having the corresponding width and height. The constraints that set the u_h variables are given by

$$\sum_{i\in I'} x_{ji\bar{e}} - |\bar{I}| + 1 \leq u_h(|I'| - |\bar{I}| + 1) \qquad \forall h \in \bar{H},\ \forall j \in J'. \tag{5.11}$$

The extended Benders cuts impose that the height of a level is at least as high as in the subproblem if all corresponding u_h variables are set to 1 and is defined as

$$\Big(\sum_{h \in \bar{H}} u_h - |\bar{H}| + 1 \Big) \bar{z} \leq z_j \qquad \forall j \in J'. \tag{5.12}$$

Moreover, we iteratively exclude the smallest item of $\bar{I}$ and resolve the subproblem as long as the objective of the optimal solution does not change. The resulting Benders cuts are in general stronger and reduce the number of master iterations.

5.1.3 Compact Formulation

The used compact formulation is straightforward, hence we omit an exact specification. The main idea is to pack items first into stacks and then pack stacks into levels. To model this, we use binary variables v_{ki} that are set to one if item i is packed into strip k and binary variables u_{jk} to express that stack k resides in level j. Each item has to be packed exactly once and each stack containing items is allowed to appear in exactly one level. Moreover, we have to ensure that the total width of all stacks belonging to the same level does not exceed W_{S}. For each of the potentially n levels an integer variable is used which has to be at least as high as the highest residing stack. Furthermore, we applied the symmetry breaking described in Section 5.1.2.

5.1.4 Computational Results

The algorithms have been implemented in C++11 and tested on an Intel Xeon E5-2630 v2, 2.60 GHz using Ubuntu 14.04. The ILP formulations have been solved with IBM ILOG CPLEX 12.6.2 using the same parameter setting as in [29]. All algorithms had a time limit of 7200 seconds. For the benchmark we use the instance sets beng, cgcut, gcut, ht and ngcut from [29].

We compare the compact formulation against our LBBD with simple Benders cuts and with extended Benders cuts. The compact model could solve 31 out of 47 test instances to optimality, which is only marginally outperformed with 32 optimally solved test instances by the LBBD using either simple or extended Benders cuts. However, the LBBD can solve some instances considerably faster as Figure 5.1 shows. For instance, after 10 seconds the LBBD with simple and with extended Benders cuts could solve 22 test instances to optimality, while the compact model could optimally solve 17 test instances.

5.1.5 Conclusions

We proposed a novel LBBD for the 3SPP and compared it with a compact formulation. The master problem relaxes the 3SPP s.t. only the first two stages of guillotine cuts are determined. The resulting subproblems are iteratively resolved to strengthen the generated Benders cuts. In addition, we proposed two kinds of Benders cuts. The experimental results have shown that the presented LBBD can solve substantially more

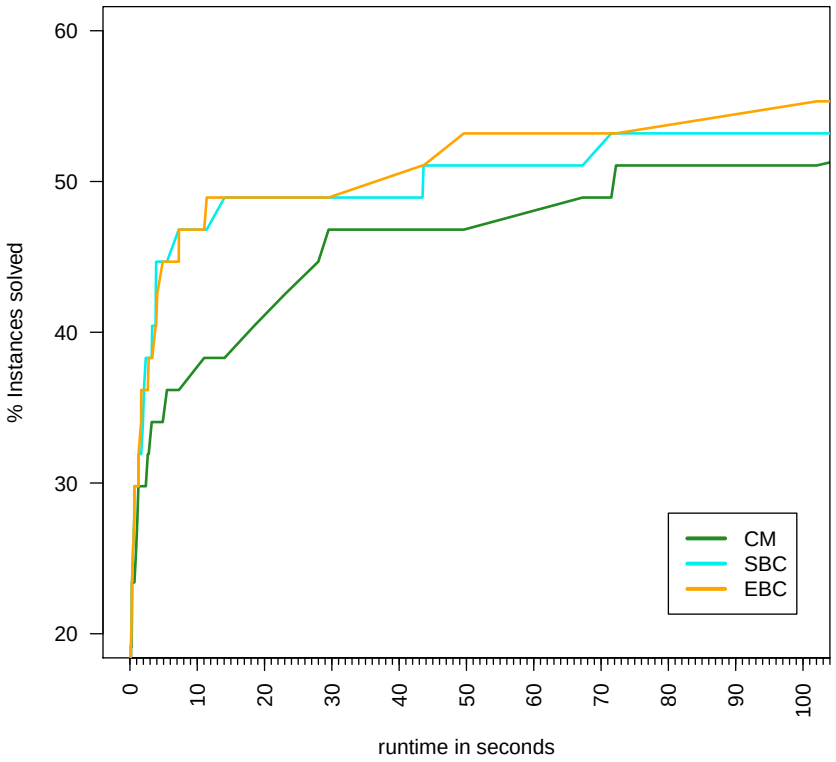


Figure 5.1: Performance profile of the first 100 seconds for the compact formulation (CM) and for the presented LBBD with simple Benders cuts (SBC) and extended Benders cuts (EBC) on the instance sets beng, cgcut, gcut, ht and ngcut from [29].

test instances in the first 100 seconds. The LBBD can solve one test instance more than the compact model within the time limit. More testing is necessary to see under which conditions the proposed approach works especially well.

5.2 High Resolution Resource-Constrained Project Scheduling

We consider a resource-constrained project scheduling problem that requires scheduling in a high temporal resolution. Traditional MILP techniques such as time-indexed formulations or discrete-event formulations are known to have severe limitations in such cases, i.e., growing too fast or having weak linear programming relaxations. We suggest a relaxation based on partitioning time into so-called time-buckets. This relaxation is iteratively solved and serves as basis for deriving feasible solutions using heuristics. The approach shows excellent performance in comparison to the traditional formulations and a metaheuristic.

5.2.1 Introduction

In job shop and project scheduling problems, generally speaking, a set of activities needs to be scheduled. The execution of the activities typically depends on certain resources of limited availability and diverse other restrictions like precedence constraints. A feasible schedule is sought that minimizes some objective function like the makespan. For such problems, MILP techniques are frequently considered, but also known to have severe limitations.

Basically, there are few general MILP modeling strategies for approaching such scheduling problems: Firstly, it is sometimes possible to come up with a compact model where the starting times of activities are directly expressed by means of corresponding variables. Resource constraints, however, impose a particular challenge in this respect. While they can be often treated in principle, e.g., by discrete-event models [53], these models are typically rather weak. A second, frequently applied option are so-called time-indexed formulations. They are based on a discretization of time, i.e., the activities may only start on a limited set of possible starting times. Binary variables are used that are additionally indexed by these possible starting times. The success of such time-indexed models strongly depends on the resolution of the time discretization. While such models can have strong LP relaxations, the number of variables and constraints increases dramatically with the number of possible starting times. Frequently, a rather crude discretization can therefore only be applied to obtain any result in reasonable computation time. 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., [53]. While they are frequently very successful, they are also substantially more complex to develop and implement.

Here, we consider a relaxation of a potentially very fine-grained time-indexed model in which the set of possible starting times is partitioned into so-called time-buckets. This time-bucket relaxation is typically much smaller than the original time-indexed model and can be solved relatively quickly. An obtained solution provides a lower bound for the time-indexed model's solution value but in general does not directly represent a feasible schedule as activity starting times are only restricted to certain time-intervals. Such solutions, however, provide a promising starting point for matheuristics. On the one hand, we may try to derive a feasible schedule by heuristically fixing the starting times to specific values, trying to fulfill all constraints. On the other hand, we can further subdivide some time-buckets and re-solve the resulting refined model to obtain an improved bound and a model that comes closer to the time-indexed model. Doing this refinement iteratively yields a matheuristic that in principle converges to a provably optimal solution. In practice, it is crucial to subdivide the time-buckets sensibly in order to increase the model's size only slowly while hopefully obtaining significantly stronger bounds.

The basic idea of the time-bucket relaxation originates in work from Wang and Regan [114] on the traveling salesman problem with time windows. Dash et al. [32] build upon

this work and suggest an iterative refinement based on the solution to the LP relaxation. We are not aware of any work that applies this principle already in the scheduling domain. There is just other work where the time-indexed model is applied with different resolutions for the time discretization, but such approaches do in general not yield lower bounds and introduce imprecisions and are therefore conceptually different.

5.2.2 Problem Definition

The above sketched general approach is more specifically investigated on a resource constrained scheduling problem with precedence constraints, which we call *simplified intraday particle therapy patient scheduling problem* (SI-PTPSP). This problem is motivated from PTPSP's time assignment subproblem of determining schedules for individual days (see Chapter 3). Following PTPSP's original definition, DTs are here modeled as a set of activities that need to be scheduled in such a way that given precedence constraints with minimum and maximum time lags are respected. Our experimental evaluation considers benchmark instances motivated by the particle therapy application.

We are given a set of resources $R = \{1, \dots, \rho\}$, a set of activities $A = \{1, \dots, \alpha\}$, and for each activity $a \in A$ a processing time p_a, a release time t^{r}_a, a deadline t^{d}_a, and a subset of required resources $Q_a \subseteq R$. Let the overall (very large) set of discrete times be $T = \{T^{\min}, \dots, T^{\max}\}$. Each resource $r \in R$ is only available at certain time intervals specified by set $W_r \subseteq T$. Last but not least, precedence constraints among the activities are stated by a directed acyclic graph $G = (A, P)$ with $P \subset A \times A$ and for each precedence relation expressed by an arc $(a, a') \in P$ minimum and maximum end-to-start time lags $L^{\min}_{a,a'}, L^{\max}_{a,a'} \in \mathbb{N}_{\geq 0}$ with $L^{\min}_{a,a'} \leq L^{\max}_{a,a'}$ need to be obeyed.

A solution $S = (S_1, \dots, S_\alpha) \in T^\alpha$ assigns to each activity $a \in A$ a starting time $S_a \in T$, from which on the activity is performed without preemption. We are looking for a feasible solution that minimizes the makespan. SI-PTPSP can be classified as $\mathrm{PS}m, \cdot, 1|r_j, d_j, temp|C_{\max}$ following the notation by Brucker et al. [19].

5.2.3 Time-Bucket Relaxation and Matheuristic

Let $B = \{B_1, \dots, B_\beta\}$ be a partitioning of T into subsequent time-buckets with $B_b = \{B^{\mathrm{start}}_b, \dots, B^{\mathrm{end}}_b\}, \forall b = 1, \dots, \beta$, and $B^{\mathrm{end}}_b + 1 = B^{\mathrm{start}}_{b+1},\ \forall b = 1, \dots, \beta - 1$. We write $I(B)$ for the index set referring to all buckets in B, i.e., $I(B) = \{1, \dots, \beta\}$. Moreover, we denote by $W^B_r(b) = |B_b \cap W_r|$ the aggregated availability of resource $r \in R$ over the whole bucket $b \in I(B)$. We refer by $C_a = \{C_{a,1}, \dots, C_{a,\gamma_a}\} \subseteq 2^{I(B)}$ to all subsets of consecutive buckets in B to which an activity $a \in A$ can be jointly assigned so that some part of activity a is performed in each of the buckets. These sets can be determined by "sliding" the activity over all time-slots and taking the covered buckets. Furthermore, let $S^{\min}_{a,c}$ be the earliest time-slots from T at which activity a can start when it is assigned to bucket sequence $C_{a,c}$, and $S^{\max}_{a,c}$ the latest. For each bucket sequence $C_{a,c} \in C_a$ and each contained bucket $b \in C_{a,c}$ we further determine a lower bound $z^{\min}_{a,b,c}$ and an upper bound

$z^{\max}_{a,b,c}$ for the number of time-slots at which activity a can take place in bucket b when activity a is assigned to $C_{a,c}$. The time-bucket relaxation can now be stated as follows:

$$\min \text{MS} \tag{5.13}$$

$$\sum_{c=1}^{\gamma_a} y_{a,c} = 1 \qquad \forall a \in A, \tag{5.14}$$

$$\sum_{c=1}^{\gamma_a} S^{\min}_{a,c} \cdot y_{a,c} + p_a \leq \text{MS} \qquad \forall a \in A, \tag{5.15}$$

$$\sum_{a \in A: r \in Q_a} \sum_{C_{a,c} \in C_a : b \in C_{a,c}} z^{\min}_{a,b,c} \cdot y_{a,c} \leq W^B_r(b) \qquad \forall r \in R,\ b \in I(B), \tag{5.16}$$

$$\sum_{c'=1}^{\gamma_{a'}} S^{\max}_{a',c'} \cdot y_{a',c'} - \sum_{c=1}^{\gamma_a} S^{\min}_{a,c} \cdot y_{a,c} \geq p_a + L^{\min}_{a,a'} \qquad \forall (a,a') \in P, \tag{5.17}$$

$$\sum_{c'=1}^{\gamma_{a'}} S^{\min}_{a',c'} \cdot y_{a',c'} - \sum_{c=1}^{\gamma_a} S^{\max}_{a,c} \cdot y_{a,c} \leq p_a + L^{\max}_{a,a'} \qquad \forall (a,a') \in P, \tag{5.18}$$

$$y_{a,c} \in \{0,1\} \qquad \forall a \in A, c = 1, \ldots, \gamma_a, \tag{5.19}$$

$$\text{MS} \geq 0. \tag{5.20}$$

Variable MS represents the makespan to be minimized (5.13). Binary variables $y_{a,c}$ indicate if activity $a \in A$ is completely performed in bucket sequence $C_{a,c}$. Equations (5.14) ensure that for each activity exactly one bucket sequence is chosen from C_a. Inequalities (5.15) are used for determining the makespan MS. Inequalities (5.16) consider for each time bucket the aggregated resource availabilities and resource consumptions for performing the respective activities. Finally, inequalities (5.17) and (5.18) represent the precedence constraints with the minimum and maximum time lags, respectively.

Our matheuristic works as follows. We initially solve the time-bucket relaxation for a rather crude partitioning of T into buckets. From the solution of the time-bucket relaxation we try to derive a feasible schedule, i.e., we try to choose valid activity starting times as far as possible in correspondence to the selected bucket sequences. This is done by an embedded GRASP metaheuristic that features two constructive heuristics and an exchange neighborhood, similar to the one used in Chapter 3. The first greedy constructive heuristic considers activities in chronological order of their latest possible finishing time within their assigned bucket, taking care of the precedence constraints and resource constraints as far as possible. Should we be able to find a feasible schedule whose makespan corresponds to the solution value of the time-bucket relaxation, then this schedule is optimal, and we can terminate. Otherwise, some activities remain unscheduled, and we apply a second heuristic to augment and repair the partial solution, possibly obtaining a feasible approximate solution and a primal bound. Here it can again be the case that we are able to close the optimality gap.

If so far no optimal solution has been found, the bucket partitioning is further refined by splitting buckets related to violated constraints. The refined model is solved again and

the whole process iterated. We investigate and compare several strategies for the bucket splitting.

5.2.4 Results and Conclusions

An experimental comparison with a compact discrete-event model and a classical time-indexed formulation clearly shows the advantages of the time-bucket relaxation: While the discrete-event model is only applicable to tiny instances due to its poor LP relaxation, the time-indexed formulation suffers from its huge size when considering practically reasonable time discretizations. The matheuristic based on the iterative refinement of the time-buckets model, however, soon yields reasonable lower bounds as well as feasible heuristic solutions, and both are improved over time. For detailed computational results see [104].

The described approach is relatively generic and can rather easily be adapted to related scheduling problems. The focus of this work was on MILP-based approaches. A promising next step would be to consider CP techniques within our matheuristic to further improve its performance. Moreover, the field of hybrid metaheuristics and matheuristics provides plenty of opportunities to further exploit the proposed time-bucket relaxation.

CHAPTER 6

Conclusions

In this thesis we considered computational methods for patient scheduling in particle therapy. In contrast to classical radiotherapy where each treatment room is equipped with an individual LINAC, in particle therapy a single synchrotron is shared between several treatment rooms. A direct consequence is that the scheduling for each treatment room cannot be done independently. Another important aspect is that the treatment rooms are occupied for some time before and after an actual irradiation of a patient. These times as well as the duration of the irradiation are strongly dependent on the applied treatment. To avoid significant breaks in the usage of the particle beam the available treatments need to be carefully arranged. Hence, the most central goal of patient scheduling in particle therapy is to maximize the throughput of such treatment centers. In an ideal schedule, the particle beam is directly switched from one treatment room to the next.

In Chapter 3 we studied the midterm planning problem arising at MedAustron, a particle therapy treatment center. We developed the PTPSP that considers a planning horizon of several months and a set of therapies each consisting of a sequence of treatments called DTs. The problem involves planning on day level as well as a dependent detailed scheduling of DTs at each day. We started by formalizing PTPSP in terms of a MILP model. However, even solving only the substructure that considers the assignment of DTs to days has shown to be computationally to difficult. Therefore, we approached PTPSP with heuristic and metaheuristic techniques. We presented a constructive heuristic that first assigns DTs therapy-wise to days and afterwards creates schedules for the individual days. This heuristic has proven to be fast and provides already reasonable results. It serves us as main building block for an initially simple IG metaheuristic. This IG has been revised and improved for several times. A main property of IG metaheuristics is that only a fraction of the incumbent solutions are destroyed and substantial information survives from one iteration to the next. In the first series of improvements we revised the destruction and construction operators with the aim to preserve the order of the

not removed DTs within the days. Compared to before more characteristics from the incumbent solution can be maintained. Moreover, a local search operator is incorporated at the end of the IG's iterations. Here, the main challenge was to find a powerful neighborhood that still can be searched fast enough to be applicable in an iterative approach. This was achieved by defining a neighborhood for each day separately and to restrict the neighborhood only to the most promising moves. The next revision considers the planning on the day level. An essential aspect in this regard is the estimation of additional used extended time resulting from assigning a considered DT to some day. To this end, we presented a surrogate model to improve this estimation for the main bottleneck resources. The finally presented revision of our IG approach addresses an extended problem statement in which the starting times for DTs belonging to the same therapy should not vary more than specified thresholds. The extension requires to additionally determine so-called nominal starting times that serve as reference point for the therapies' DTs. As the DT's starting times and the nominal starting times are strongly-dependent, our approach is to determine and update them alternatingly. For our computational study we generated a set of benchmark instances related to the expected situation at MedAustron. Our first experiments indicated that our initial IG is superior to a GRASP. We concluded that for instances of practically relevant size it is of utmost importance to reuse information from an already optimized incumbent solution in order to obtain a well performing approach. For the proposed changes of our IG we showed that each of them provide a substantial improvement on our instances.

Our experiments also acted as a proof of concept, but there still remain challenges. It can be expected that our artificial benchmark instances do not entirely reflect all aspects of the future practice. Our algorithm configurations are, however, based to some degree on the made assumptions. A promising way to improve the applicability of our approach is to define a parameter model that determines a suitable algorithm configuration on basis of the observed instance characteristics. In this thesis we focused on the treatment appointments of the therapies directly involving the particle beam. There are, however, also other activities which should preferably be included in the optimization as well. These include a treatment planning phase preceding all DTs. In this stage personalized equipment needed for irradiation is produced. Moreover, a substantial amount of planning has to be done to determine the detailed treatment strategy. These activities underlie other constraints and cannot be modeled as DTs. For accompanying these surrounding activities we suggest customized post-processing techniques.

In Chapter 4 the PC-JSOCMSR problem is considered that has the characteristics of particle therapy scenario but also appears in other contexts. We studied the application of DDs for this problem. Our DDs are based on a recursive model for PC-JSOCMSR. In contrast to the problems already considered with DDs in the literature, solutions for PC-JSOCMSR do not consist of a fixed number of elements. First, we adapted TDC and IR, the two traditional compilation techniques for DDs, to PC-JSOCMSR. To the best of our knowledge, we were the first to directly compare both methods experimentally. The strict layer orientation of these traditional methods represents,

however, a substantial limitation and yields redundancies in the obtained DDs. For this reason, we propose a novel compilation method for relaxed DDs based on principles of A* search. The construction is driven by a quickly obtainable upper bound and merges similar states across layers. The latter is accomplished by labeling the corresponding nodes in state-space-relaxation fashion. Nevertheless, layer-oriented merging can easily be achieved for problems where it is natural. Furthermore, we propose a construction method for restricted DDs that exploits information on promising feasible solutions encoded in a relaxed DDs. We conducted our experiments on benchmark instances based on the particle therapy scenario as well as on an avionics setting. Our results for the two conventional, layer-oriented compilations methods indicated that depending on the instance class the IR provides either similar or stronger bounds. For the small to medium-sized instances the restricted DDs constructed by the TDC yielded better bounds than a DD-independent GRASP. The constructed relaxed DDs by our A*-based approach are frequently an order of magnitude smaller than those compiled with the TDC and the derived upper bounds are most of the times substantially better. The successive compilation of a restricted MDD based on a relaxed MDD resulted in a substantial speedup. With our proposed compilation methods we are usually able to construct a relaxed DD and a restricted DD in less time than the conventional TDC requires for a restricted DD of the same quality and size. In doing so, we are not only able to derive frequently stronger primal bounds, but at the same time also provide dual bounds. Finally, we showed that our novel approaches are also advantageous in comparison to a GRASP, an MILP, and a CP approach for instances up to 500 jobs. Both of the newly proposed compilation approaches appear highly promising also for other problems.

With PC-JSOCMSR, we studied a problem where the typically layered structure of DD is not natural and leads to redundancies. The flexibility of our A*-based approach has been the key to clearly outperform existing compilation techniques. In future work it would be of interest to study whether the A*-search principles are also beneficial for problems where the cardinality solutions is fixed and the layer is included in the label. In this case, merging based on a label that also includes a fast-to-calculate upper bound seems in particular interesting. Although our approach yields a primal and a dual bound it is not per se a complete approach, meaning that it is not guaranteed to reach a proven optimal solution as soon as nodes are merged. A further next step would be to embed our DD into an algorithmic framework that converges to an optimal solution. We demonstrated that the information encoded in DDs can be successfully exploited. Another interesting topic is to explore whether metaheuristic methods can benefit from a given relaxed DD and might further offer new possibilities for DD compilation techniques.

In Chapter 5, we considered SI-PTPSP, a RCPSP with precedence constraints that requires scheduling in a high temporal resolution. The problem is again motivated by our patient scheduling application. Instead of modeling DTs by jobs with varying resource requirements, sequences of activities linked by minimum and maximum time lags are used. Our proposed matheuristic repeatedly solves and refines a so-called time-bucket relaxation. The approach is relatively generic and should be promising also for related

scheduling problems. Especially the proposed time-bucket relaxation provides many opportunities for further successful exploitation.

Last but not least, we studied in Chapter 5 also the 3SPP, a strip-packing variant restricted to three stages of guillotine cuts. The presented LBBD considers in its master problem only the first two stages, while the third stage is solved in the subproblems. The experimental results have shown that the LBBD outperforms a corresponding compact MILP model. A promising next step is to hybridize the approach with heuristic or metaheuristic techniques.

In this thesis, we demonstrated that patient scheduling in particle therapy is not only practically relevant but is also algorithmically an interesting and challenging problem. While the consideration of the whole planning horizon and all problem-specific details are already demanding for heuristic and metaheuristic techniques, substructures of the problem are more general and suitable for advanced approaches. Therefore, we did not only develop a solution method for a single problem appearing in the real world but gained insights that are applicable also for other problems.

APPENDIX A

Prize-Collecting Job Sequencing: Referred Methods

For the sake of completeness we describe here the upper bound calculation and reference approaches used in Chapter 4 that have been developed by my coauthors and are published in [56]. We start in Section A.1 with details regarding the used upper bound calculation for PC-JSOCMSR. Afterwards, in Section A.2 and Section A.3 we summarize the MILP and CP approaches that are referred in the computational study of Chapter 4.

A.1 Calculation of Upper Bound $Z^{\text{ub}}(u)$

We adopt in the A*-based construction of relaxed DDs the upper bound calculation from Horn et al. [56]. For a given node u with state $(P(u), t(u))$, an upper bound for the still achievable total prize by the remaining jobs in $P(u)$ can be calculated by solving the following LP relaxation of a multi-constrained 0–1 knapsack problem.

$$Z^{\text{ub}}_{\text{KP}}(u) = \max \sum_{j \in P(u)} z_j x_j \tag{A.1}$$

$$\text{s.t.} \sum_{j \in P(u)} p^0_j x_j \leq W_0(P(u), t(u)), \tag{A.2}$$

$$\sum_{j \in P(u) \cap J_r} p_j x_j \leq W_r(P(u), t(u)) \qquad r \in R, \tag{A.3}$$

$$0 \leq x_j \leq 1 \qquad j \in P(u). \tag{A.4}$$

Here, x_j is a continuous relaxation of a binary variable that indicates if job $j \in P(u)$ is scheduled or not. The right-hand-sides of the knapsack constraints are

$$W_0(P(u), t(u)) = \left| \bigcup_{j \in P(u)} \bigcup_{\substack{w=1,\dots,\omega_j: \\ W_{jw}^{\text{end}} - p_j^{\text{post}} \geq t_0(u) + p_j^0}} \left[\max(t_0(u), W_{jw}^{\text{start}} + p_j^{\text{pre}}), W_{jw}^{\text{end}} - p_j^{\text{post}} \right] \right| \tag{A.5}$$

and

$$W_r(P(u), t(u)) = \left| \bigcup_{j \in P(u) \cap J_r} \bigcup_{w=1,\dots,\omega_j : W_{jw}^{\text{end}} \geq t_r(u) + p_j} \left[\max(t_r(u), W_{jw}^{\text{start}}), W_{jw}^{\text{end}} \right] \right|. \tag{A.6}$$

They represent the total amount of still available time for resource 0 and resource $r \in R$, respectively, considering the current state and the time windows.

Solving this LP model exactly within our A*-based construction is, however, computationally too expensive. Instead, we compute upper bounds by solving two types of relaxations. The first one is obtained by relaxing inequalities (A.3):

$$Z_0^{\text{ub}}(u) = \max \sum_{j \in P(u)} z_j x_j \tag{A.7}$$

$$\text{s.t.} \sum_{j \in P(u)} p_j^0 x_j \leq W_0(P(u), t(u)), \tag{A.8}$$

$$0 \leq x_j \leq 1 \qquad j \in P(u). \tag{A.9}$$

The second relaxation is obtained by performing a Lagrangian relaxation of inequality (A.2), where $\lambda \geq 0$ is the Lagrangian dual multiplier associated with inequality (A.2):

$$h^{\text{ub}}(u, \lambda) = \max \sum_{j \in P(u)} z_j x_j + \lambda \left(W_0(P(u), t(u)) - \sum_{j \in P} p_j^0 x_j \right) \tag{A.10}$$

$$\text{s.t.} \sum_{j \in P(u) \cap J_r} p_j x_j \leq W_r(P(u), t(u)) \qquad r \in R, \tag{A.11}$$

$$0 \leq x_j \leq 1 \qquad j \in P(u). \tag{A.12}$$

Here, J_r denotes again the subset of all jobs in J which require secondary resource $r \in R$. Both, $Z_0^{\text{ub}}(u)$ and $h^{\text{ub}}(u, \lambda)$, are computed by solving LP relaxations of simple knapsack problems. In the latter case, this is possible since the problem separates over the resources and for each resource, the resulting problem is an LP relaxation of a knapsack problem. An LP relaxation of a knapsack problem can be efficiently solved by a greedy algorithm that packs items in decreasing prize/time-ratio order. The first item that does not completely fit is packed partially so that the capacity is exploited as far as possible, see [66].

It follows from weak duality (see e.g. [88], Prop. 6.1) that $h^{\text{ub}}(u, \lambda)$ yields an upper bound on $Z^{\text{ub}}_{\text{KP}}(u)$ for all $\lambda \geq 0$, but the quality of this upper bound depends on the choice of λ. We consider $h^{\text{ub}}(u, \lambda)$ for the values $\lambda = 0$ and $\lambda = z_{\bar{j}}/p^0_{\bar{j}}$, where $\bar{j}$ denotes the last, and typically partially, packed item in an optimal solution to the problem solved to obtain $Z^{\text{ub}}_0(u)$. The value $\lambda = z_{\bar{j}}/p^0_{\bar{j}}$ is chosen since it is an optimal LP dual solution associated with inequality (A.8) and therefore has a chance to be a good estimate of a value for λ that gives a strong upper bound.

For our A*-based construction of relaxed DDs we use the strongest bound on $Z^{\text{ub}}_{\text{KP}}(u)$ that we can obtain by solving the relaxations introduced above:

$$Z^{\text{ub}}(u) = \min\left\{Z^{\text{ub}}_0(u), h^{\text{ub}}(u, 0), h^{\text{ub}}(u, z_{\bar{j}}/p^0_{\bar{j}})\right\}. \tag{A.13}$$

A.2 Order-Based Mixed Integer Linear Programming Model

Horn et al. [56] proposed the following order-based MILP model for PC-JSOCMSR. It uses binary variable t_j to indicate if job $j \in J$, is included in the schedule or not and binary variable t_{jw} to indicate if job $j \in J$ is assigned to time window $w = 1, \ldots, \omega_j$ or not. Let the continuous variable s_j be the start time of job j. Binary variable $y_{jj'}$ is further used to indicate if job j is scheduled before j' w.r.t. the common resource or not, if both jobs are scheduled, $j, j' \in J,\ j \neq j'$. For each job $j \in J$, let the release time be $T^{\text{rel}}_j = min_{\omega=1,\ldots,\omega_j} W^{\text{start}}_{j\omega}$, and let the deadline be $T^{\text{dead}}_j = \max_{\omega=1,\ldots,\omega_j} W^{\text{end}}_{j\omega}$, $j \in J$.

$$\max \sum_{j \in J} z_j t_j \tag{A.14}$$

$$\text{s.t. } t_j = \sum_{w=1,\ldots,\omega_j} t_{jw} \qquad j \in J, \tag{A.15}$$

$$y_{jj'} + y_{j'j} \geq t_j + t_{j'} - 1 \qquad j, j' \in J,\ j \neq j', \tag{A.16}$$

$$s_{j'} \geq s_j + \delta_{jj'} - (T^{\text{dead}}_j - p_j - T^{\text{rel}}_{j'} + \delta_{jj'})(1 - y_{jj'}) \qquad j, j' \in J,\ j \neq j', \tag{A.17}$$

$$s_j \geq T^{\text{rel}}_j + \sum_{w=1,\ldots,\omega_j} \left(W^{\text{start}}_{jw} - T^{\text{rel}}_j\right) t_{jw} \qquad j \in J, \tag{A.18}$$

$$s_j \leq T^{\text{dead}}_j - p_j + \sum_{w=1,\ldots,\omega_j} (W^{\text{end}}_{jw} - T^{\text{dead}}_j) t_{jw} \qquad j \in J, \tag{A.19}$$

$$t_j \in \{0, 1\} \qquad j \in J, \tag{A.20}$$

$$t_{jw} \in \{0, 1\} \qquad w = 1, \ldots, \omega_j,\ j \in J, \tag{A.21}$$

$$T^{\text{rel}}_j \leq s_j \leq T^{\text{dead}}_j - p_j \qquad j \in J, \tag{A.22}$$

$$y_{jj'} \in \{0, 1\} \qquad j, j' \in J,\ j \neq j'. \tag{A.23}$$

Equations (A.15) state that each scheduled job must be assigned to a time window and inequalities (A.16) ensure that if two jobs j and j' are scheduled, either $y_{jj'}$ or $y_{j'j}$ must

be set to one, i.e., one of them needs to precede the other. If a job is to precede another one, inequalities (A.17) ensure this w.r.t. the jobs' start times. Here, $\delta_{jj'}$ is the minimum time between the start of job j and the start of job j' if job j is scheduled before job j', which depends on whether both jobs use the same resource or not, i.e.,

$$\delta_{jj'} = \begin{cases} p_j, & \text{if } q_j = q_{j'}, \\ p_j^{\text{pre}} + p_j^0 - p_{j'}^{\text{pre}}, & \text{if } q_j \neq q_{j'}. \end{cases} \tag{A.24}$$

If a job is assigned to a time window, inequalities (A.18) and (A.19) make its start time comply with this time window. Otherwise, the job only has to comply with its release time and deadline.

A.3 Constraint Programming Model

Horn et al. [56] also proposed the following CP model for PC-JSOCMSR, which makes use of so-called *option type* variables. Such a variable may either have a value of a certain domain assigned or set to the special value $\top$ that indicates the absence of a value. For job $j \in J$ we use the option type variable s_j for the job's start time. An absent start time, i.e., $s_j = \top$, indicates that the job is not scheduled.

$$\max \sum_{j \in J \mid \text{occurs}(s_j)} z_j \tag{A.25}$$

$$\text{disjunctive\_strict}(\{(s_j + p_j^{\text{pre}}, p_j^0) \mid j \in J\}) \tag{A.26}$$

$$\text{disjunctive\_strict}(\{(s_j, p_j) \mid j \in J \wedge q_j = r\}) \qquad r \in R \tag{A.27}$$

$$\text{occurs}(s_j) \implies \left(s_j \in \bigcup_{\omega=1,\ldots,\omega_j} \left[W_{j\omega}^{\text{start}}, W_{j\omega}^{\text{end}} - p_j \right] \right) \qquad j \in J \tag{A.28}$$

$$s_j \in [T_j^{\text{rel}}, \ldots, T_j^{\text{dead}} - p_j] \cup \{\top\} \qquad j \in J \tag{A.29}$$

For job $j \in J$ the predicate $\text{occurs}(s_j)$ yields true if the option type variable s_j is not absent, i.e., job j is scheduled. The strict disjunctive constraints (A.26) and (A.27) ensure that all scheduled jobs do not overlap w.r.t. their usage of the common resource and the secondary resource $r \in R$, respectively. Absent jobs are hereby ignored. Constraints (A.28) state that if job $j \in J$ is scheduled, it must be performed within one of the job's time windows.

Acronyms

3SPP 3-staged strip packing problem.

BDD binary decision diagram.

COP combinatorial optimization problem.

CP constraint programming.

DD decision diagram.

DP dynamic programming.

DT daily treatment.

GA genetic algorithm.

GRASP greedy randomized adaptive search procedure.

GVNS general variable neighborhood search.

IG iterated greedy.

ILP integer linear program.

ILS iterated local search.

IR incremental refinement.

IRLP incremental refinement guided by longest paths.

JSOCMSR job sequencing with one common and multiple secondary resources.

LBBD logic-based Benders decomposition.

LINAC linear accelerator.

LP linear program.

MDD multivalued decision diagram.

MILP mixed integer linear program.

PC-JSOCMSR prize-collecting job sequencing with one common and multiple secondary resources.

PFSP permutation flow shop problem.

PTPSP particle therapy patient scheduling problem.

RCPSP resource-constrained project scheduling problem.

SI-PTPSP simplified intraday particle therapy patient scheduling problem.

SPP strip packing problem.

TDC top-down construction.

TWCH therapy-wise construction heuristic.

VND variable neighborhood descent.

VNS variable neighborhood search.

Bibliography

[1] S. B. Akers. Binary decision diagrams. *IEEE Transactions on Computers*, 27(6): 509–516, 1978.

[2] A. Allahverdi. A survey of scheduling problems with no-wait in process. *European Journal of Operational Research*, 255(3):665–686, 2016.

[3] H. R. Andersen, T. Hadzic, J. N. Hooker, and P. Tiedemann. A constraint store based on multivalued decision diagrams. In *Principles and Practice of Constraint Programming – CP 2007*, pages 118–132. Springer Berlin Heidelberg, 2007.

[4] B. S. Baker, J. E. G. Coffman, and R. L. Rivest. Orthogonal packings in two dimensions. *SIAM Journal on Computing*, 9(4):846–855, 1980.

[5] R. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, 1957.

[6] J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4(1):238–252, 1962.

[7] D. Bergman and A. A. Cire. Theoretical insights and algorithmic tools for decision diagram-based optimization. *Constraints*, 21(4):533–556, 2016.

[8] D. Bergman and A. A. Cire. On finding the optimal BDD relaxation. In *Integration of Constraint Programming, Artificial Ingelligence and Operations Research, CPAIOR 2017*, volume 10335 of *LNCS*, pages 41–50. Springer, 2017.

[9] D. Bergman, W.-J. van Hoeve, and J. N. Hooker. Manipulating MDD relaxations for combinatorial optimization. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 20–35. Springer Berlin Heidelberg, 2011.

[10] D. Bergman, A. A. Cire, W.-J. van Hoeve, and J. N. Hooker. Optimization bounds from binary decision diagrams. *INFORMS Journal on Computing*, 26(2):253–268, may 2014.

[11] D. Bergman, A. A. Cire, W.-J. van Hoeve, and T. Yunes. BDD-based heuristics for binary optimization. *Journal of Heuristics*, 20(2):211–234, 2014.

[12] D. Bergman, A. A. Cire, W.-J. van Hoeve, and J. N. Hooker. *Decision Diagrams for Optimization*. Artificial Intelligence: Foundations, Theory, and Algorithms. Springer, 2016.

[13] D. Bergman, A. A. Cire, W.-J. van Hoeve, and J. N. Hooker. Discrete optimization with decision diagrams. *INFORMS Journal on Computing*, 28(1):47–66, 2016.

[14] M. Blikstad, E. Karlsson, T. Lööw, and E. Rönnberg. An optimisation approach for pre-runtime scheduling of tasks and communication in an integrated modular avionic system. *Optimization and Engineering*, 19(4):977–1004, 2018.

[15] C. Blum and G. R. Raidl. *Hybrid Metaheuristics*. Springer International Publishing, 2016.

[16] F. Bray, A. Jemal, N. Grey, J. Ferlay, and D. Forman. Global cancer transitions according to the human development index (2008–2030): a population-based study. *The Lancet Oncology*, 13(8):790–801, 2012.

[17] F. Bray, J. Ferlay, I. Soerjomataram, R. L. Siegel, L. A. Torre, and A. Jemal. Global cancer statistics 2018: GLOBOCAN estimates of incidence and mortality worldwide for 36 cancers in 185 countries. *CA: A Cancer Journal for Clinicians*, 68(6):394–424, 2018.

[18] P. Brucker and S. Knust. Resource-constrained project scheduling and timetabling. In *Practice and Theory of Automated Timetabling III*, volume 2079 of *LNCS*, pages 277–293. Springer, 2001.

[19] 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.

[20] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, 1986.

[21] E. K. Burke, P. L. Rocha, and S. Petrovic. An integer linear programming model for the radiotherapy treatment scheduling problem. *CoRR*, abs/1103.3391, 2011.

[22] N. Christofides, A. Mingozzi, and P. Toth. State-space relaxation procedures for the computation of bounds to routing problems. *Networks*, 11:145–164, 1981.

[23] A. A. Cire and W. V. Hoeve. Multivalued decision diagrams for sequencing problems. *Operations Research*, 61(6):1411–1428, 2013.

[24] D. Conforti, F. Guerriero, and R. Guido. Optimization models for radiotherapy patient scheduling. *4OR*, 6(3):263–278, 2008.

[25] D. Conforti, F. Guerriero, and R. Guido. Non-block scheduling with priority for radiotherapy treatments. *European Journal of Operational Research*, 201(1): 289–296, 2010.

[26] D. Conforti, F. Guerriero, R. Guido, and M. Veltri. An optimal decision-making approach for the management of radiotherapy patients. *OR Spectrum*, 33(1): 123–148, 2011.

[27] M. Conforti, G. Cornuéjols, and G. Zambelli. *Integer Programming*. Springer International Publishing, 2014.

[28] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing – STOC '71*. ACM Press, 1971.

[29] J.-F. Côté, M. Dell'Amico, and M. Iori. Combinatorial benders' cuts for the strip packing problem. *Operations Research*, 62(3):643–661, 2014.

[30] G. B. Dantzig. Maximization of a linear function of variables subject to linear inequalities. *Activity Analysis of Production and Allocation*, pages 339–347, 1951.

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

[32] 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.

[33] M. Dorigo and T. Stützle. Ant colony optimization: Overview and recent advances. In *Handbook of Metaheuristics*, pages 311–351. Springer International Publishing, 2018.

[34] T. A. Feo and M. G. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8(2):67–71, 1989.

[35] T. A. Feo and M. G. C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6(2):109–133, 1995.

[36] M. L. Fisher. The lagrangian relaxation method for solving integer programming problems. *Management Science*, 27(1):1–18, 1981.

[37] M. R. Garey and D. S. Johnson. *Computers and Intractability, a Guide to the Theory of NP-Completness*. WH Freeman and Co, San Francisco, 1979.

[38] M. Gendreau and J.-Y. Potvin. Tabu search. In *Handbook of Metaheuristics*, pages 41–59. Springer US, 2010.

[39] M. Gendreau and J.-Y. Potvin. *Handbook of metaheuristics*, volume 2. Springer, 2010.

[40] P. C. Gilmore and R. E. Gomory. Sequencing a one-state variable machine: A solvable case of the traveling salesman problem. *Operations Research*, 12(5):655–679, 1964.

[41] Y. Gocgun. Simulation-based approximate policy iteration for dynamic patient scheduling for radiation therapy. *Health Care Management Science*, 21(3):317–325, 2016.

[42] R. Graham, E. Lawler, J. Lenstra, and A. Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Discrete Optimization II*, volume 5 of *Annals of Discrete Mathematics*, pages 287–326. Elsevier, 1979.

[43] M. Guignard and S. Kim. Lagrangean decomposition for integer programming : theory and applications. *RAIRO - Operations Research - Recherche Opérationnelle*, 21(4):307–323, 1987.

[44] T. Hackl. Local search methods for the particle therapy patient scheduling problem. Master's thesis, TU Wien, Vienna, Austria, 2018.

[45] T. Hadzic, J. N. Hooker, B. O'Sullivan, and P. Tiedemann. Approximate compilation of constraints into multivalued decision diagrams. In *Lecture Notes in Computer Science*, pages 448–462. Springer Berlin Heidelberg, 2008.

[46] P. Hansen and N. Mladenović. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130(3):449–467, 2001.

[47] P. Hansen, N. Mladenović, J. Brimberg, and J. A. M. Pérez. Variable neighborhood search. In *Handbook of Metaheuristics*, pages 61–86. Springer US, 2010.

[48] P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4 (2):100–107, 1968.

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

[50] M. Hifi. Exact algorithms for the guillotine strip cutting/packing problem. *Computers & Operations Research*, 25(11):925–940, 1998.

[51] S. Hoda, W.-J. van Hoeve, and J. N. Hooker. A systematic approach to MDD-based constraint programming. In *Principles and Practice of Constraint Programming – CP 2010*, pages 266–280. Springer Berlin Heidelberg, 2010.

[52] J. Hooker and G. Ottosson. Logic-based benders decomposition. *Mathematical Programming*, 96(1):33–60, 2003.

[53] J. N. Hooker. Planning and scheduling by logic-based benders decomposition. *Operations Research*, 55(3):588–602, 2007.

[54] J. N. Hooker. Decision diagrams and dynamic programming. In *CPAIOR 2013: Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 7874 of *LNCS*, pages 94–110. Springer, 2013.

[55] M. Horn, G. R. Raidl, and C. Blum. Job sequencing with one common and multiple secondary resources: A problem motivated from particle therapy for cancer treatment. In *The Third International Conference on Machine learning, Optimization and big Data – MOD 2017*, volume 10710 of *Lecture Notes in Computer Science*. Springer International Publishing, 2017.

[56] M. Horn, G. Raidl, and E. Rönnberg. An A* algorithm for solving a prize-collecting sequencing problem with one common and multiple secondary resources and time windows. In *PATAT 2018: Proceedings of the 12th International Conference of the Practice and Theory of Automated Timetabling*, pages 235–256, Vienna, Austria, 2018.

[57] M. Horn, J. Maschler, G. R. Raidl, and E. Rönnberg. A*-based construction of decision diagrams for a prize-collecting scheduling problem. *INFORMS Journal on Computing*, submitted.

[58] A. J. Hu. *Techniques for Efficient Formal Verification Using Binary Decision Diagrams*. PhD thesis, Stanford University, Stanford, CA, USA, 1995.

[59] M. Höfler. Methods for intraday scheduling in particle therapy. Master's thesis, TU Wien, Vienna, Austria, 2018.

[60] L. W. Jacobs and M. J. Brusco. A local-search heuristic for large set-covering problems. *Naval Research Logistics*, 42(7):1129–1140, 1995.

[61] T. Jatschka. An iterative refinement algorithm for high resolution scheduling problems. Master's thesis, TU Wien, Vienna, Austria, 2017.

[62] T. Kapamara and D. Petrovic. A heuristics and steepest hill climbing method to scheduling radiotherapy patients. In *Proceedings of the International Conference on Operational Research Applied to Health Services (ORAHS)*, Leuven, Belgium, 2009. Catholic University of Leuven.

[63] T. Kapamara, K. Sheibani, O. Haas, D. Petrovic, and C. Reeves. A review of scheduling problems in radiotherapy. In *Proceedings of the International Control Systems Engineering Conference (ICSE)*, pages 201–207, Coventry, UK, 200. Coventry University Publishing.

[64] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984.

[65] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Springer US, 1972.

[66] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, 2004.

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

[68] J. Kinable, A. A. Cire, and W. J. van Hoeve. Hybrid optimization methods for time-dependent sequencing problems. *European Journal of Operational Research*, 259(3):887–897, 2017.

[69] J. Kleinberg and E. Tardos. *Algorithm design*. Pearson, 2006.

[70] S. N. Larsson. Radiotherapy patient scheduling using a desktop personal computer. *Clinical Oncology*, 5(2):98–101, 1993.

[71] C. Y. Lee. Representation of switching circuits by binary-decision programs. *Bell System Technical Journal*, 38(4):985–999, 1959.

[72] A. Legrain, M.-A. Fortin, N. Lahrichi, and L.-M. Rousseau. Online stochastic optimization of radiotherapy patient scheduling. *Health Care Management Science*, 18(2):110–123, 2015.

[73] P. Leite Rocha. *Novel Approaches to Radiotherapy Treatment Scheduling*. PhD thesis, University of Nottingham, 2011.

[74] A. Lodi, S. Martello, and D. Vigo. Models and bounds for two-dimensional level packing problems. *Journal of Combinatorial Optimization*, 8(3):363–379, 2004.

[75] 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.

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

[77] J. Maschler and G. R. Raidl. A logic-based Benders decomposition approach for the 3-staged strip packing problem. In *Operations Research Proceedings 2015*, pages 393–399. Springer, 2017.

[78] J. Maschler and G. R. Raidl. Particle Therapy Patient Scheduling with Limited Starting Time Variations of Daily Treatments. *International Transactions in Operational Research*, 2018.

[79] J. Maschler and G. R. Raidl. Multivalued decision diagrams for a prize-collecting sequencing problem. In *Proceedings of the 12th International Conference of the Practice and Theory of Automated Timetabling*, pages 375–397, Vienna, Austria, 2018.

[80] J. Maschler and G. R. Raidl. Multivalued decision diagrams for prize-collecting job sequencing with one common and multiple secondary resources. *Annals of Operations Research*, submitted.

[81] J. Maschler, M. Riedler, M. Stock, and G. R. Raidl. Particle therapy patient scheduling: First heuristic approaches. In *Proceedings of the 11th International Conference on the Practice and Theory of Automated Timetabling*, pages 223–244, Udine, Italy, 2016.

[82] J. Maschler, T. Hackl, M. Riedler, and G. R. Raidl. An enhanced iterated greedy metaheuristic for the particle therapy patient scheduling problem. In *Proceedings of the 12th Metaheuristics International Conference*, pages 465–474, Barcelona, Spain, 2017.

[83] J. Maschler, M. Riedler, and G. R. Raidl. Particle therapy patient scheduling: Time estimation to schedule sets of treatments. In *Extended Abstracts of the 16th International Conference on Computer Aided Systems Theory (EUROCAST 2017)*, pages 106–107, Gran Canaria, Spain, 2017.

[84] J. Maschler, M. Riedler, and G. R. Raidl. Particle therapy patient scheduling: Time estimation for scheduling sets of treatments. In *Computer Aided Systems Theory – EUROCAST 2017*, volume 10671 of *Lecture Notes in Computer Science*, pages 364–372. Springer International Publishing, 2018.

[85] C. Men. *Optimization models for radiation therapy: treatment planning and patient scheduling*. PhD thesis, University of Florida, 2009.

[86] N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100, 1997.

[87] M. Nawaz, E. E. Enscore, and I. Ham. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 11(1):91–95, 1983.

[88] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, 1988.

[89] A. G. Nikolaev and S. H. Jacobson. Simulated annealing. In *Handbook of Metaheuristics*, pages 1–39. Springer US, 2010.

[90] P. S. Ow and T. E. Morton. Filtered beam search in scheduling. *International Journal of Production Research*, 26(1):35–62, 1988.

[91] Q.-K. Pan and R. Ruiz. An effective iterated greedy algorithm for the mixed no-idle permutation flowshop scheduling problem. *Omega*, 44:41–50, 2014.

[92] C. H. Papadimitriou and K. Steiglitz. *Combinatorial optimization: Algorithms and complexity*. Dover Publications, New York, 1998.

[93] D. Petrovic, M. Morshed, and S. Petrovic. Genetic algorithm based scheduling of radiotherapy treatments for cancer patients. *Proceedings of the Conference on Artificial Intelligence in Medicine (AIME)*, 5651:101–105, 2009.

[94] D. Petrovic, M. Morshed, and S. Petrovic. Multi-objective genetic algorithms for scheduling of radiotherapy treatments for categorised cancer patients. *Expert Systems with Applications*, 38(6):6994–7002, 2011.

[95] S. Petrovic and P. Leite-Rocha. Constructive Approaches to Radiotherapy Scheduling. In *World Congress on Engineering and Computer Science (WCECS)*, pages 722–727, 2008.

[96] S. Petrovic and P. Leite-Rocha. Constructive and GRASP approaches to radiotherapy treatment scheduling. In *Advances in Electrical and Electronics Engineering - IAENG Special Edition of the World Congress on Engineering and Computer Science 2008*, pages 192–200. IEEE, 2008.

[97] S. Petrovic, W. Leung, X. Song, and S. Sundar. Algorithms for radiotherapy treatment booking. In *25th Workshop of the UK Planning and Scheduling Special Interest Group*, pages 105–112, Nottingham, UK, 2006.

[98] J. Puchinger and G. R. Raidl. Models and algorithms for three-stage two-dimensional bin packing. *European Journal of Operational Research*, 183(3): 1304–1327, 2007.

[99] S. F. Rad, R. Ruiz, and N. Boroojerdian. New high performing heuristics for minimizing makespan in permutation flowshops. *Omega*, 37(2):331–345, apr 2009.

[100] G. R. Raidl, T. Jatschka, M. Riedler, and J. Maschler. Time-bucket relaxation based mixed integer programming models for scheduling problems: A promising starting point for matheuristics. In *Proceedings of Matheuristics 2016: 6th International Workshop on Model-Based Metaheuristics*, pages 104–107, Brussels, Belgium, 2016.

[101] C. R. Reeves. Genetic algorithms. In *Handbook of Metaheuristics*, pages 109–139. Springer US, 2010.

[102] M. G. Resende and C. C. Ribeiro. Greedy randomized adaptive search procedures: Advances, hybridizations, and applications. In *Handbook of Metaheuristics*, pages 283–319. Springer US, 2010.

[103] M. Riedler. *Advances in Decomposition Approaches for Mixed Integer Linear Programming*. PhD thesis, TU Wien, Vienna, Austria, 2018.

[104] 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*, 2017.

[105] M. Römer, A. A. Cire, and L.-M. Rousseau. A local search framework for compiling relaxed decision diagrams. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research, CPAIOR 2018*, volume 10848 of *LNCS*, pages 512–520. Springer, 2018.

[106] F. Rossi, P. Van Beek, and T. Walsh. *Handbook of constraint programming*. Elsevier, 2006.

[107] R. Ruiz and T. Stützle. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177(3):2033–2049, 2007.

[108] A. Sauré, J. Patrick, S. Tyldesley, and M. L. Puterman. Dynamic multi-appointment patient scheduling for radiation therapy. *European Journal of Operational Research*, 223(2):573–584, 2012.

[109] B. W. Stewart and C. P. Wild. *World Cancer Report 2014*. World Health Organization, 2014.

[110] J. A. A. Van der Veen, G. J. Wöginger, and S. Zhang. Sequencing jobs that require common resources on a single machine: A solvable case of the TSP. *Mathematical Programming*, 82(1-2):235–254, 1998.

[111] B. Vieira, E. W. Hans, C. van Vliet-Vroegindeweij, J. van de Kamer, and W. van Harten. Operations research for resource planning and -use in radiotherapy: a literature review. *BMC Medical Informatics and Decision Making*, 16(1), 2016.

[112] P. Vogl, R. Braune, and K. F. Doerner. A multi-encoded genetic algorithm approach to scheduling recurring radiotherapy treatment activities with alternative resources, optional activities, and time window constraints. In *Computer Aided Systems Theory – EUROCAST 2017*, volume 10671 of *Lecture Notes in Computer Science*, pages 373–382. Springer International Publishing, 2018.

[113] P. Vogl, R. Braune, and K. F. Doerner. Scheduling recurring radiotherapy appointments in an ion beam facility. *Journal of Scheduling*, 2018.

[114] 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.

[115] WHO. The top 10 causes of death. http://www.who.int/mediacentre/factsheets/fs310/en/, May 2018. Accessed: 2018-12-13.

Curriculum Vitae

Personal information

Surname / First name	**Maschler, Johannes**
Address	Mollardgasse 18/1/14, 1060 Vienna, Austria
Telephone	+43 677 61762347
Email	johannes.maschler@gmail.com
Date of birth	December 1, 1988

Education

TU Wien, Vienna, Austria
since 2014 **Doctoral programme in Engineering Sciences**
Field of Study Computer Science
Thesis *Patient Scheduling in Particle Therapy*
Advisor Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Günther R. Raidl

TU Wien, Vienna, Austria
2011–2013 **Master of Science**
Field of Study Software Engineering & Internet Computing
Thesis *Data Exchange of Relational Data and Beyond*
Advisor Univ.Prof. Mag.rer.nat. Dr.techn. Reinhard Pichler

TU Wien, Vienna, Austria
2007–2011 **Bachelor of Science**
Field of Study Medical Informatics

Professional Activities

2014–2015 **University Assistant**
Algorithms and Complexity Group
Institute of Logic and Computation

2015–2018 **Project Assistant**
Research collaboration on patient scheduling with EBG MedAustron GmbH, Wiener Neustadt, Austria

Publications

2018 Matthias Horn, Johannes Maschler, Günther R. Raidl, and Elina Rönnberg.
A*-based construction of decision diagrams for a prize-collecting scheduling problem.
INFORMS Journal on Computing, submitted

Johannes Maschler and Günther R. Raidl.
Multivalued decision diagrams for prize-collecting job sequencing with one common and multiple secondary resources.
Annals of Operations Research, submitted

Johannes Maschler and Günther R. Raidl.
Multivalued decision diagrams for a prize-collecting sequencing problem.
In *Proceedings of the 12th International Conference of the Practice and Theory of Automated Timetabling*, pages 375–397, Vienna, Austria, 2018

Johannes Maschler and Günther R. Raidl.
Particle Therapy Patient Scheduling with Limited Starting Time Variations of Daily Treatments.
International Transactions in Operational Research, 2018

Johannes Maschler, Martin Riedler, and Günther R. Raidl.
Particle therapy patient scheduling: Time estimation for scheduling sets of treatments.
In *Computer Aided Systems Theory – EUROCAST 2017*, volume 10671 of *Lecture Notes in Computer Science*, pages 364–372. Springer International Publishing, 2018

2017 Johannes Maschler, Thomas Hackl, Martin Riedler, and Günther R. Raidl.
An enhanced iterated greedy metaheuristic for the particle therapy patient scheduling problem.
In *Proceedings of the 12th Metaheuristics International Conference*, pages 465–474, Barcelona, Spain, 2017

Martin Riedler, Thomas Jatschka, Johannes Maschler, and Günther R. Raidl.
An iterative time-bucket refinement algorithm for a high-resolution resource-constrained project scheduling problem.
International Transactions in Operational Research, 2017

Johannes Maschler and Günther R. Raidl.
A logic-based Benders decomposition approach for the 3-staged strip packing problem.
In *Operations Research Proceedings 2015*, pages 393–399. Springer, 2017

2016 Johannes Maschler, Martin Riedler, Markus Stock, and Günther R. Raidl.
Particle therapy patient scheduling: First heuristic approaches.
In *Proceedings of the 11th International Conference on the Practice and Theory of Automated Timetabling*, pages 223–244, Udine, Italy, 2016