universität wien

DISSERTATION

Titel der Dissertation

An Application of Constraint Programming and Boolean Satisfiability Solving Techniques to Variants of the Resource-Constrained Project Scheduling Problem

verfasst von

Alexander Schnell, Bakk.rer.nat. MSc

angestrebter akademischer Grad

Doctor of Philosophy (PhD)

Wien, 2015

Studienkennzahl lt. Studienblatt:	A 094 151 403
Dissertationsgebiet lt. Studienblatt:	Betriebswirtschaft (Logistics and Operations Management)
Betreut von:	o. Univ.-Prof. Dipl.-Ing. Dr. Richard F. Hartl

Acknowledgements

Firstly, I would like to thank my advisor, Prof. Richard F. Hartl for having given me the possibility to work on this interesting topic. I deeply acknowledge that Prof. Hartl always gave me the necessary space to conduct the research for this thesis. Furthermore, I am highly thankful that Prof. Hartl always supported me with his scientific knowledge, both regarding the theoretic and the strategic aspects. Prof. Hartl always took his time when I had questions and provided greatly valuable input. Furthermore, I am grateful that he always motivated me and backed my will to continue on the chosen topic.

Secondly, I am very grateful for the possibility to have worked in such a nice research environment. Therefore, I cannot thank my former and present colleagues at the Chair of Production and Operations Management enough for always having open doors and ears when it comes to talking about scientific and non-scientific topics. You really motivated me and influenced my attitude in a positive way. I have learned much from you all.

Moreover, I want to thank the anonymous reviewers of my papers which I submitted throughout the years. They always provided highly valuable and constructive comments which greatly improved the theory and results of this dissertation. I am happy that it is like that in our research area. Furthermore, I would like to convey my thanks to the people which I met at conferences. I really appreciated the possibility to discuss the topics of my dissertation and other research topics with persons from many different countries. Furthermore, I would like to express my thanks to the SCIP team. Their optimization software and the respective support is great. They always reacted quickly to my questions on the mailing list and their input was a great help when I had problems during the coding phases.

I am also very grateful to my parents, Hans and Ute, for constantly supporting me throughout the years. They always treasured my dissertation plan which strengthened my confidence during this phase. Finally, I am highly thankful to Sonja for keeping the things together at home and for accepting my moods during the hard phases of this dissertation. Her positive thinking and cordial and warm manner supported me very much on my way to realize this dissertation.

Preface

This dissertation contains the main results of my research during the last four and a half years at the University of Vienna. The research conducted for this dissertation was a constant process of developing and evaluating new solution concepts. As a consequence, many solution concepts which looked nice in theory did not pass the computational evaluations in practice and are therefore omitted in the following.

Parts of this dissertation are also published as extended abstracts in conference proceedings or are under revision in international journals:

- A preliminary version of Chapt. 5 is published as an extended abstract in the conference proceedings of the International Conference on Project Management and Scheduling 2014 (see [71]).
- An extended abstract based on Chapt. 7 is part of the proceedings of the MISTA conference 2013 [67].
- Full papers based on the Chapters 5 and 6 are under revision in international journals (see [69] and [70]).
- Chapter 4 also exists as a working paper (see [68]).

Vienna, February 2015 Alexander Schnell

Contents

Abbreviations **1**

1 Introduction **3**
1.1 Solution approaches 4
1.2 Structure of the thesis 5

2 The resource-constrained project scheduling problem (RCPSP) **7**
2.1 Problem description 7
2.1.1 Project and resource environment 7
2.1.2 Job characteristics 8
2.1.3 Objective functions 10
2.2 General problem formulation 12
2.3 Complexity results 13
2.4 Preprocessing techniques 14
2.5 An example for the multi-mode resource-constrained project scheduling problem (MRCPSP) with generalized precedence relations (GPRs) 17

3 Constraint Programming (CP) and Boolean Satisfiability (SAT) Solving for the RCPSP **21**
3.1 Introduction to CP 21
3.1.1 Preliminaries 21
3.1.2 CP for scheduling problems 28
3.2 Introduction to SAT Solving 34
3.2.1 Conflict Analysis 35
3.2.2 Conflict-based variable-value selection strategies and restarts 41
3.3 Combination of CP and SAT techniques to solve the RCPSP 41
3.3.1 Lazy clause generation 42
3.3.2 The optimization framework SCIP 45
3.3.3 Conclusion on optimization frameworks 46

4 Performance variability of CP-SAT solvers for the RCPSP **47**
4.1 A CP model for the single-mode resource-constrained project scheduling problem (SRCPSP) with standard precedence relations (SPRs) . 47

4.2 Activity List Preprocessing and Independent Parallel Solution 48
4.3 Computational Experiments 49
4.3.1 Positive Effects through the Generation of more Activity Lists 51
4.3.2 Comparisons to Recent General Exact Approaches 55
4.3.3 Effect of a different branching rule 56
4.4 Conclusion 56

5 CP-SAT approaches for the MRCPSP with SPRs 61
5.1 G12- and SCIP-models for the MRCPSP with SPRs 61
5.1.1 A formulation for the Zinc-modeling language 61
5.1.2 Formulations for SCIP 63
5.2 Principles of the `cumulativemm`-constraint 64
5.2.1 Explanations for timetable propagation (TP) with multi-mode jobs 66
5.2.2 Comparison to other explanation generation techniques and possible improvements 68
5.3 Computational experiments 71
5.4 Conclusion 74

6 CP-SAT approaches for the MRCPSP with GPRs 77
6.1 Introduction 77
6.2 A SCIP-model for the MRCPSP with GPRs 77
6.2.1 Strengthening of the mathematical formulation 78
6.3 Key Principles of `sprecedencemm` 81
6.3.1 Domain Propagation 81
6.3.2 Explanation Generation 82
6.4 Key Principles of `gprecedencemm` 83
6.5 Computational Experiments 84
6.6 Conclusion 89

7 A heuristic for the multi-mode resource-constrained multi-project scheduling problem (MRCMPSP) with SPRs 91
7.1 A SCIP-model for the MRCMPSP with SPRs 91
7.2 Local Search for the MRCMPSP with SPRs 93
7.3 Results 94
7.4 Conclusion 94

8 Conclusion and future research 97

Bibliography 103

List of Figures 113

List of Tables 115

List of Algorithms **117**

Abstract **119**

Zusammenfassung **121**

Curriculum Vitæ **123**

Abbreviations

AL activity list

SRCPSP single-mode resource-constrained project scheduling problem

MRCPSP multi-mode resource-constrained project scheduling problem

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

RCPSP resource-constrained project scheduling problem

COPs combinatorial optimization problems

RCPS resource-constrained project scheduling

SPRs standard precedence relations

GPRs generalized precedence relations

CP Constraint Programming

CSP constraint satisfaction problem

COP constraint optimization problem

SAT Boolean Satisfiability

AoN Activity-on-Node

MIP Mixed-Integer Programming

TPD total project delay

TMS total makespan

CA Conflict Analysis

BaB Branch and Bound

LCG lazy clause generation

MPI minimal problem instance

BCI bound change index

NPV *net present value*

RPC *resource-profile cost*

LCG lazy clause generation

MPV minimal processing version

SSGS serial schedule generation scheme

BCP backtracking with constraint propagation

TP timetable propagation

UP unit propagation

CA conflict analysis

UIP unique implication point

VSIDS variable state independent decaying sum

EF edge finding

DPLL Davis–Putnam–Logemann–Loveland

ER energetic reasoning

Chapter 1

Introduction

The aim of resource-constrained project scheduling (RCPS) is to assign starting times to a number of jobs subject to precedence and resource constraints such that a project related objective is optimized. RCPS belongs to the operational level of project management [43]. In practice it is important to avoid the "escalation" of a project "in time and budget" (cf. [43, p.27]). The approaches of commercial software packages to tackle project scheduling problems are based on simple priority rules [43]. The application of the latter can lead to highly suboptimal solutions for the underlying problem class. Moreover, for more complex problem variants it can even be hard to find feasible solutions with these techniques. Thus, it is important to investigate more sophisticated approaches for RCPS to overcome the above disadvantages.

Variants of the RCPSP arise in a wide range of applications [36]. Scientific RCPS techniques are e.g. applied to plan the production and engineering in the automotive industry [9], to schedule IT projects [40] and to schedule the outbound baggage at international airports (see [30]).

In this dissertation, the main research topic is the implementation and analysis of new exact approaches combining CP and SAT Solving techniques for the SRCPSP and the MRCPSP with SPRs and GPRs. These problems are known to be highly complex combinatorial optimization problems (COPs).

Exact approaches for COPs find the optimal solution w.r.t. a certain objective function and prove its optimality for a finite input within a finite number of iterations. The main bottleneck of exact approaches is the high computational time, they need for the solution of large instances of complex COPs. Therefore, in practice heuristics are preferred to find good solutions in acceptable computational times without proving their optimality.

Nevertheless, exact approaches are of high relevance, as they evaluate lower and upper bounds for the underlying problem and thus can provide a quality measure for problem-specific heuristics. Moreover, they can be applied for the solution of subproblems in hybrid metaheuristics [14] to tackle larger problem instances in practice.

1.1 Solution approaches

Exact algorithms for variants of the SRCPSP can be divided into procedures applying general and procedures applying specialized solution concepts. These algorithms are on both sides Branch and Bound (BaB)-based at which the rules for branching, bounding, backtracking and the fathoming of nodes differ. Thereby, the specialized algorithms integrate principles specially tailored to the variant of the SRCPSP at hand. On the other hand, the general solution procedures employ a mixture of concepts from CP, SAT Solving and Mixed-Integer Programming (MIP) which are generalizable to other problem classes.

The state-of-the-art specialized algorithms for the SRCPSP with SPRs and the aim of makespan minimization are among others the ones by Demeulemeester and Herroelen [25] and Sprecher [79] whereas the latter works with a significantly lower memory usage[1].

Recently, it has turned out that also SAT Solving techniques can be successfully applied for the solution of variants of the SRCPSP. Horbach [44] proposes a standalone SAT Solving approach for the SRCPSP with SPRs and the aim of makespan minimization which is based on *minimal forbidden sets*. His extensive computational experiments show that his approach is highly competitive to chosen exact algorithms considered as state-of-the-art in the literature. Furthermore, his approach outperforms all of them on certain instance sets from the PSPLIB [50].

Moreover, the experiments of Schutt et al. [76, 77, 75] show, that the combination of CP and SAT Solving techniques leads to additional improvements. They applied lazy clause generation (LCG) (a BaB approach integrating CP and SAT solving techniques [28]) to the SRCPSP with SPRs and GPRs and the aim of makespan minimization, and to the SRCPSP with SPRs and discounted cash flows, respectively. Their computational results show that LCG is at the moment the best exact approach for the latter variants of the SRCPSP.

Berthold et al. [11] additionally integrate MIP techniques for the solution of the SRCPSP with SPRs and the aim of makespan minimization. It turns out, that their combination of CP, SAT Solving and MIP techniques is inferior to LCG regarding solution time and quality. Nevertheless, an improvement of the lower bounds for some instances from the literature is reported.

Regarding exact approaches for the SRCPSP with SPRs, it is also important to mention the work of Koné et al. [51]. They compare different MIP formulations for the latter problem and test their efficiency with a state-of-the-art MIP solver.

An overview on heuristic approaches for the SRCPSP with SPRs and GPRs is given in [49] or [3], and [77], respectively.

Specialized exact approaches for the MRCPSP with SPRs have been summarized and tested by Hartmann and Drexl [37], whereas they conclude that the approach of Sprecher and Drexl [81] is the exact method of choice. The most recent exact

[1] Detailed surveys on exact algorithms for the SRCPSP are given by Demeulemeester and Herroelen [26] and Brucker and Knust [16].

algorithm of Zhu et al. [92] outperforms the latter approach. They implemented a Branch-and-Cut procedure with a preprocessing and a heuristic step to generate good upper bounds as an input for their algorithm.

A recent survey on heuristic approaches for the MRCPSP with SPRs and a detailed experimental evaluation is given by [87]. Their computational experiments show that the scatter search procedure of Van Peteghem and Vanhoucke [86] produces the best results. In this context, it is also important to mention the approach of Coelho and Vanhoucke [18] as they combine SAT solving techniques with a metaheuristic for the SRCPSP to solve the MRCPSP with SPRs.

De Reyck and Herroelen [24] were the first to consider the MRCPSP with GPRs. Thereafter, Heilmann [39] developed an exact approach for the latter problem, which is to our knowledge at the moment still state-of-the-art. He proposed a specialized BaB-algorithm based on the solution of a "minimal problem instance" and branched on the mode alternative or renewable resource conflict resolution which is the hardest w.r.t. a specific measure. Lower bounds for the MRCPSP with GPRs have been evaluated by Brucker and Knust [15].

The procedure of Ballestín et al. [7], a combination of simulated annealing and an evolutionary algorithm, and the tabu search approach of Nonobe and Ibaraki [61] are state-of-the-art heuristics for the MRCPSP with GPRs.

1.2 Structure of the thesis

This dissertation begins with two introductory chapters to provide the basics about the RCPSP and the theory of CP and SAT Solving to the reader.

Chapter 2 contains a problem description and classification of RCPSP variants relevant for this dissertation. Moreover, it discusses computational complexity results and well-known preprocessing techniques from the literature. The chapter ends with an example for the MRCPSP with GPRs and a possible solution of the latter.

Chapter 3 outlines the theory of CP and SAT Solving. Furthermore, it describes how these two solution principles can be combined to tackle RCPSP instances.

After the introductory part, we summarize our contribution to the research area of project scheduling.

Chapter 4 discusses a factor leading to a significant performance variability in recent CP-SAT approaches for the SRCPSP with SPRs. We detected a great variation in the efficiency of these algorithms depending on the predefined search space given through the initial variable domains. Moreover, the search space leading to the best results is in most cases not the one defined through the smallest variable domains. A detailed computational analysis of this effect is presented, also w.r.t. an explanation of the latter observation. Furthermore,

as the variation of the predefined search space has highly positive effects on the efficiency of the analyzed algorithms, we develop a parallel procedure to take advantage of these effects.

The main contribution of this thesis is the extension of state-of-the-art CP-SAT approaches for the SRCPSP to the MRCPSP.

Chapter 5 proposes three CP models for the MRCPSP with SPRs which can be formulated in optimization frameworks that integrate an exact solution approach combining CP and SAT Solving techniques. For one modeling formulation, we implemented a new global constraint `cumulativemm` within SCIP [2] specially tailored to renewable resources in the context of multi-mode jobs. The best of our solution approaches outperforms the state-of-the-art exact approach of Zhu et al. [92]. Moreover, we close (find the optimal solution and prove its optimality for) 324 open problem instances from the literature.

Chapter 6 proposes new CP-SAT based modeling and solution approaches for the MRCPSP with GPRs extending the ones presented in Chapter 5. Therefore, we implemented two new constraint handlers `sprecedencemm` and `gprecedencemm` for SCIP. They guarantee feasibility w.r.t. SPRs and GPRs in the context of multi-mode activities, respectively. Furthermore, they capture domain propagation and explanation generation algorithms for the latter problem characteristics. We show how our SCIP-models can be strengthened by integrating `sprecedencemm` and `gprecedencemm`, i.e. by adding problem-specific knowledge about precedence relations. With our best performing solution approach, we close 289 open instances and improve the best known makespan for 271 instances from the literature.

Chapter 7 shows how easily our modeling and solution concepts can be extended to more complex problem classes. In this chapter, we present a SCIP-model for the MRCMPSP with SPRs which extends our model for the MRCPSP with SPRs. This chapter was motivated by the MISTA 2013 challenge [90] where the aim was to solve instances of the latter problem in an efficient way. Small challenge instances can be solved exactly with SCIP. To tackle larger problem instances, we propose a local search method based on the iterative solution of subproblems by SCIP. Overall, our solution procedure was ranked ninth among all competition participants.

The dissertation ends with a conclusion on the obtained results and a discussion of further research topics.

Chapter 2

The RCPSP

2.1 Problem description

In [17], the classification scheme for machine scheduling [35] and resource-constrained machine scheduling [13] was extended to classify variants of the RCPSP. In the following, we introduce the problems tackled in this dissertation based on a modified version of the $[\alpha|\beta|\gamma]$-classification scheme in [17]. In this context, α, β and γ characterize the project and resource environment, the jobs, and the objective function, respectively.

Note that, we only discuss the characteristics of project scheduling problems important for our applications. For other possible variants the reader is referred to the survey of Hartmann and Briskorn [36].

2.1.1 Project and resource environment

By integrating the abbreviation *PS* into the α-classification, it is specified, that variants of the RCPSP are considered.

The classical RCPSP integrates a set of renewable resources $R = \{1, \ldots, p_\rho\}$ and a set of nonrenewable resources $N = \{1, \ldots, p_\nu\}$. A limited capacity C_r^ρ of the renewable resource $r \in R$ is available in every time interval $[t, t+1[$ of the planning horizon. In practice, this can i.e. correspond to a fixed number of workers or machines available in every time period of the planning horizon. C_k^ν denotes the available amount of the nonrenewable resource $r \in N$ for the complete project planning horizon $[0, T[$. A fixed budget or an amount of raw material for the project can be an example of a nonrenewable resource in practice.

For the classification of the resource environment, the notation $res^\nu n_1, n_2, n_3$ and $res^\rho r_1, r_2, r_3$ is used to denote nonrenewable and renewable resources, respectively. The specification of numbers $r_1 \in \mathbb{N}$ and $r_2,\ r_3 \in \mathbb{R}$ means that we consider a problem with $|R| = r_1$ renewable resources with the capacity $r_2 = C_r^\rho,\ \forall r \in R$ and each activity consumes at most r_3 units of every renewable resource.

If r_1, r_2 or r_3 is substituted by $\cdot$, we consider the problem where the respective parameter is arbitrary, i.e. it is defined by the input. If r_3 is replaced by $\cdot$, we omit it in the classification. Similarly if r_2 and r_3, or r_1 and r_2 and r_3 are all replaced by $\cdot$, we leave out all the respective parameters. For example, if we consider a project scheduling problem with p_ρ renewable resources, where the renewable resource capacities and the resource consumption of the activities are part of the input, α corresponds to $PS, res^\rho r$. Nonrenewable resources are denoted in a similar way. If selecting the resource amount is part of the decision, we add ∞ to the α-classification.

2.1.2 Job characteristics

A project consists of a set of activities or jobs $J = \{1, \ldots, n\}$. We only consider problems where the jobs cannot be preempted. So once a job is started, it cannot be stopped before reaching its complete duration and rescheduled later. If preemption is allowed we insert *pmtn* for β.

In real-life projects, there are often various alternatives (modes) in which the activities can be processed. This corresponds to tradeoffs between processing time and resource consumption (time-resource) and tradeoffs between the consumption of different resources (resource-resource) for an activity [27]. Consider for example the activity "Debugging of a plugin" in the project "Development of a new software". If two specialists process the activity, it will approximately take two weeks. Alternatively, if one specialist and three qualified interns are assigned, they will approximately debug for three weeks. This reflects the modes and the alternative duration and resource consumption of an activity.

In this case, we introduce a set of alternatives or modes $M_i = \{1, \ldots, m_i\}$ for the processing of job $i \in J$,. If there is at least one job $i \in J$ with $|M_i| > 1$, a multi-mode problem is considered and we add *MM* to the $\beta-$classification. Otherwise, in case of a single-mode problem, the latter characterization is omitted.

Every multi-mode job's duration d_i, nonrenewable and renewable resource consumption $c^\nu_{i,r}$ and $c^\rho_{i,r}$ of resource r equals the duration $d_{i,k}$ and resource consumption $c^\nu_{i,k,r}$ and $c^\rho_{i,k,r}$ corresponding to the mode $k \in M_i$ chosen for job i. Note, that the consideration of nonrenewable resources is only relevant in a multi-mode situation. In concrete, for variants of the SRCPSP, it can be checked beforehand, if $\sum_{i \in J} c^\nu_{i,r} \leq C^\nu_r, \ r \in N$, i.e. if the nonrenewable resource capacities are respected by the input data. Thus, w.l.o.g. for variants of the SRCPSP, the notation $res^\rho \ n_1, n_2, n_3$ can be omitted.

In this dissertation, we only consider deterministic durations. In case, the durations are stochastic, we add $d_j = stoch$ to the $\beta-$classification. In addition, we assume that all parameters concerning the durations and resource consumptions are part of the input. If for example the special case would be considered, where all duration are equal to one, the notation $d_j = 1$ is inserted.

Note, that $c^\nu_{i,k,r}$ corresponds to the consumption of job i of the nonrenewable resource $r \in N$ if processed in mode k. I.e. if we choose mode k for job i, after its

processing $C_r^\nu - c_{i,k,r}^\nu$ units remain of the resource $r \in N$ for the complete project. $c_{i,k,r}^\rho$ denotes the renewable resource consumption of job i in mode k from resource $r \in R$ per period of its processing time.

Schedules for a project often have to comply with precedence relations between the jobs. For non-preemptive jobs, we distinguish between SPRs and GPRs. denoted by *prec* and *temp* in the $\beta-$classification. A SPR (i, j) between the single-mode jobs i and j forces job j to start after the end of job i. If s_i and s_j are the starting times of job i and j, respectively, the following equation has to be fulfilled:

$$s_i + d_i \leq s_j$$

With GPRs, more general situations can be modeled. Every GPR (i, j) comes with a so-called *minimal* or *maximal timelag* $d_{i,j}^{\min}$ or $d_{i,j}^{\max}$ dependent on both jobs in the precedence relation. In case of a minimal timelag $d_{i,j}^{\min}$, job j must start after job i has been processed for $d_{i,j}^{\min}$ time units, i.e.:

$$s_i + d_{i,j}^{\min} \leq s_j$$

With a maximal timelag $d_{i,j}^{\max}$ at hand, job j must start before job i has been processed for $d_{i,j}^{\max}$ time units, i.e.:

$$s_i + d_{i,j}^{\max} \geq s_j \iff s_j - d_{i,j}^{\max} \leq s_i$$

Thus, we can transform every GPR (i, j) with the maximal timelag $d_{i,j}^{\max} \geq 0$ to the GPR (j, i) with the minimal timelag $-d_{i,j}^{\max}$. Hence, w.l.o.g. it is sufficient to only consider GPRs with possibly negative minimal timelags $\delta_{i,j}$. Additionally, every type of precedence relation between non-preemptive jobs (start-start, start-finish, finish-start and finish-finish) can be transformed to an instance with GPRs and possibly negative timelags $\delta_{i,j}$ [10]. Consequently, with problems of the type *temp* in the $\beta-$classification, one can model quite general situations in practice.

Let us now again look at the above example project "Development of a new software" to illustrate a practical application of maximal time lags. Assume that a specialist from another company has to process the activity "Development of plugin X" and he is only available until a certain deadline after its completion. Then, the successor activity "Integration of plugin X into software" should be completed before the availability deadline of the specialist. This can be modelled by a maximal time lag between the predecessor and successor activity.

In case of a multi-mode instance, we have minimal timelags $\delta_{i,j,k,l} \in \mathbb{R}$ for every GPR (i, j) and the respective modes $k \in M_i$ and $l \in M_j$. Hence, in this situation, the timelags do not only depend on the jobs in the precedence relation but also on the mode k and l chosen for the job i and j, respectively.

The precedence graph can be visualized through a so-called Activity-on-Node (AoN) network. For single-mode problems the nodes correspond to jobs. The arcs connect nodes for jobs which form a precedence relation. Furthermore, the arcs

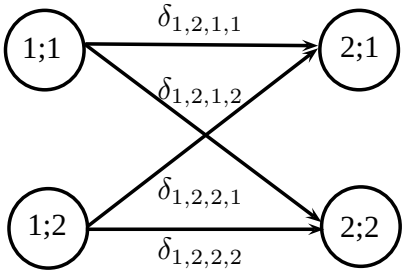

Figure 2.1: Example: GPR between multi-mode jobs

are weighted by d_i and $\delta_{i,j}$ for SPRs and GPRs, respectively. With a multi-mode instance at hand, the nodes correspond to job-mode combinations $i;k$. Two nodes are connected if a precedence relation has to be respected between the corresponding jobs. Moreover, the arc weights are the durations $d_{i,k}$ or the minimal timelags $\delta_{i,j,k,l}$ in case of SPRs and GPRs, respectively (See Fig. 2.1).

In the following, we will use a set of successors $\mathfrak{S}_i$ covering all jobs $j \in J$ forming a precedence relation with job i for the formulation of the precedence constraints. Moreover, for the sake of simplicity we assume, that the durations $d_{i,k}$ and renewable resource consumptions $c^{\rho}_{i,k,r}$, C^{ρ}_r are natural numbers for all $i \in J$, $k \in M_i$ and $r \in R$, and that the minimal timelags $\delta_{i,j,k,l}$ are integers for all $i, j \in J$ and the respective modes k and l. All the presented approaches in this dissertation can be generalized to the case where the problem parameters are element of $\mathbb{Q}$ or $\mathbb{R}$. In every case, all the approaches have to cope with numerical issues.

2.1.3 Objective functions

Well-known objective functions in the context of project scheduling, similar to the ones in machine scheduling, are the project duration or makespan $D_{\max}$, the maximum lateness L_{max} of the jobs, the weighted completion time $\sum_{j \in J} w_j \cdot (s_j + d_j)$ etc.. Note, that if we omit the parameter γ in the classification, we only consider the problem of finding a feasible schedule w.r.t. the given α- and β-classification. Further objective functions like for example the *net present value* (NPV), or the *resource-profile cost* (RPC) have also been extensively studied in the literature [36].

The net present value is defined as

$$\text{NPV} = \sum_{j \in J} C_j^f \cdot \mu^{s_j + d_j} \tag{2.1}$$

A constant $C_j^f \in \mathbb{R}$ is assigned to every job j. C_j^f is discounted with the rate $0 < \mu < 1$ depending on the completion time $s_j + d_j$ of job j. μ can for example integrate an inflation rate.

A constant $C_j^f < 0$ corresponds to a cost, i.e. we have to pay an amount of money at the completion of the job j (cash outflow) and $C_j^f > 0$ corresponds to a revenue, i.e. we receive money at its completion (cash inflow). With the NPV, the present value of C_j^f when finishing job j at time $s_j + d_j$ is approximated. The present value of the amount $|C_j^f|$ decreases with an increasing completion time t. Roughly speaking, if a job is associated with a cash inflow (outflow), one would try to complete it as soon (late) as possible. The aim is to maximize the NPV, i.e. to maximize the total present value over all jobs.

The RPC, which has to be minimized, is given as follows:

$$\text{RPC} = \sum_{r \in R} K_r \cdot f(k_r(\mathbf{s}, 0), \ldots, k_r(\mathbf{s}, T-1)) \tag{2.2}$$

K_r is a cost associated with the renewable resource $r \in R$ and $T \in \mathbb{N}$ is an upper bound on the project duration.

$k_r(\mathbf{s}, t)$ is defined as the resource usage of resource r in the schedule $\mathbf{s} \in \mathbb{R}^n$ in the time interval $[t, t+1[$. The function $f : \mathbb{R}^T \to \mathbb{R}$ assigns a value to the resource profile $(k_r(\mathbf{s}, 0), \ldots, k_r(\mathbf{s}, T-1))$.

If the RPC is considered as objective, apart from adding the latter abbreviation to the γ-classification, α integrates ∞ to denote that determining the resource usage is part of the decision. Project scheduling problems integrating RPC are also known as resource leveling problems [58]. For example the project scheduling problem, where $f(\ldots)$ corresponds to $\max_{t \in \{0,\ldots,T-1\}} k_r(\mathbf{s}, t)$, is called the resource investment problem.

In this dissertation, we mainly consider the project makespan $D_{\max} = \max_{j \in J}(s_j + d_j)$ as objective. In this situation, it is convenient to introduce dummy jobs 0 and $n+1$ and insert them to the set J. The dummy jobs are used to guarantee that the project starts at time point ≥ 0 and to model the project end. For multi-mode problems, the dummy jobs $j \in \{0, n+1\}$ can only be processed in one mode with $d_{j,1} = 0$ and $c^{\nu}_{j,1,r} = 0,\ r \in N (c^{\rho}_{j,1,r} = 0,\ r \in R)$.

Furthermore, we have to assure that no job starts before job 0 and ends after the start of job $n+1$. Therefore, for problems with SPRs we introduce a new precedence relation $(0, j)$ with the timelags 0 for every job-mode combination $j;k$ if job j has no predecessor. Moreover, if job i has no successor, we introduce the precedence relation $(i, n+1)$ with the positive timelags $d_{i,k},\ k \in M_i$.

In the presence of GPRs more care has to be taken. In this case, we introduce a new precedence relation $(0, j)$ with the timelags $\delta_{0,j,1,l} = 0$ for every job j and the corresponding modes $l \in M_j$. Moreover, we define a new precedence relation $(i, n+1)$ with the non-negative timelags $\delta_{i,n+1,k,1} = d_{i,k}$ for every job i and the corresponding modes $k \in M_i$.

After the addition of the above data, minimizing $D_{\max}$ is equal to minimizing the starting time of s_{n+1}. Note, that the introduction of dummy jobs for single-mode problem instances is based on the same principles .

In this dissertation, we mainly focus on exact solution approaches for the problems $[PS, res^{\rho}|prec|D_{\max}]$, $[PS, res^{\rho}, res^{\nu}|MM, prec|D_{\max}]$ and $[PS, res^{\rho}, res^{\nu}|MM, temp|D_{\max}]$. As they all integrate the same objective function, we will denote them in the following as the SRCPSP with SPRs, and the MRCPSP with SPRs and GPRs, respectively.

2.2 General problem formulation

The most general problem considered in this dissertation is the MRCPSP with GPRs. In the following, we only provide a problem formulation for the above problem, as all other analyzed problems are special cases of the latter.

Note, that we assume that for the following model, all the parameters, sets and variables have been updated w.r.t. the principles applied in the last section for the introduction of the dummy jobs 0 and $n+1$.

With the notation from the last section and the integral decision variables s_i and x_i corresponding to the starting time and the mode choice of job $i \in J$, respectively, the MRCPSP with GPRs can be formulated as follows:

$$\min \quad s_{n+1} \tag{2.3}$$

$$s.t. \quad s_i + \delta_{i,j,x_i,x_j} \leq s_j, \quad \forall i \in J,\ \forall j \in \mathfrak{S}_j \tag{2.4}$$

$$\sum_{j \in J} c^{\nu}_{j,x_j,r} \leq C^{\nu}_r, \qquad \forall r \in N \tag{2.5}$$

$$\sum_{j \in \mathcal{A}(t)} c^{\rho}_{j,x_j,r} \leq C^{\rho}_r, \quad \forall t \geq 0, \forall r \in R$$

$$\mathcal{A}(t) = \{j \in J : t - d_{j,x_j} + 1 \leq s_j \leq t\}, \quad \forall t \in \mathbb{N} \tag{2.6}$$

$$s_j \in \mathbb{N}, \quad \forall j \in J \tag{2.7}$$

$$x_j \in M_j, \quad \forall j \in J \tag{2.8}$$

Our aim is the minimization of the makespan $D_{\max}$. This is equal to minimizing the starting time s_{n+1}.

(2.4) model GPRs for the multi-mode jobs. The successor job $j \in \mathfrak{S}_i$ must start after the starting time of job i plus the possibly negative minimal timelag δ_{i,j,x_i,x_j} which depends on the modes x_i and x_j chosen for the respective jobs. In case of

SPRs, it holds that:

$$\delta_{i,j,x_i,x_j} = d_{i,x_i}, \quad \forall i \in J,\ j \in \mathfrak{S}_j \tag{2.9}$$

In other words, in the context of SPRs, the minimal timelags which have to be respected for the precedence relation (i,j) only depend on the duration d_{i,x_i} of the mode in which i is processed.

(2.5) guarantees that the total nonrenewable resource consumption of $r \in N$ does not exceed the available capacity C_r^ν.

With (2.6), we model the renewable resource constraints. We have to respect the limit of C_r^ρ units for all $r \in R$ in every time interval $[t, t+1[,\ t \in \mathbb{N}$. The total consumption of the renewable resource $r \in R$ over all jobs $j \in \mathcal{A}(t)$, i.e. active in the time interval $[t, t+1[$, must not exceed C_r^ρ. Note, that this formulation would lead to an infinite number of constraints, but we can restrict ourselves to values $t \in \{0, \ldots, T_{\max} - 1\}$, where:

$$T_{\max} = \sum_{i \in J} \max\{\max\{d_{i,k} : k \in M_i\}, \max\{\delta_{i,j,k,l} : j \in \mathfrak{S}_i,\ k \in M_i,\ l \in M_j\}\}, \tag{2.10}$$

This is due to the fact, that if an optimal solution $\mathbf{s}^*$ for the above problem exists, it holds that $s_{n+1}^* \leq T_{\max}$. Clearly, $T_{\max}$ is also an upper bound for the variables $s_i,\ i \in J$.

Note, that if we omit the mode choice, i.e. the variables x_j in (2.8), and the respective indices in all parameters of the model, we obtain the SRCPSP with GPRs and as a special case the SRCPSP with SPRs.

The above mathematical formulation is not directly transferable to a MIP or CP solver. Firstly, it contains variables as indices of parameters. This modeling technique is usually not available in MIP solvers but normally in CP modeling languages like e.g. in Zinc [54] or JaCoP [46] (See also Chapter 5). Secondly, in (2.6), we sum over all $j \in \mathcal{A}(t)$. The set $\mathcal{A}(t)$ itself depends on the variables s_j and x_j. This adds nonlinearity to the model and can usually only be tackled by transformations.

The well-known MIP-formulation (See e.g. [50]) of the MRCPSP with GPRs is based on the ideas of Pritsker et al. [65]. They were the first to provide an efficient binary integer programming formulation for project scheduling problems which is extendable to model the above problem.

In the Chapters 4, 5 and 6, we show how the formulation of the SRCPSP with SPRs, the MRCPSP with SPRs and the MRCPSP with GPRs can be realized in selected CP-modeling languages.

2.3 Complexity results

The MRCPSP with GPRs can be divided into a mode-assignment and a single-mode scheduling step [18]. The mode-assignment step is given through the feasibility problem with the constraint (2.5) and the variables $x_j \in M_j$. Thus, we have to find processing modes for every job $j \in J$ such that the capacity limits of the

nonrenewable resources $r \in N$ are not exceeded. With a feasible mode-assignment $\bar{\mathbf{x}}$ at hand, one has to solve a SRCPSP instance with GPRs or as a special case the SRCPSP with SPRs. The complete problem consists of finding a feasible mode-assignment at which the minimal makespan of the resulting SRCPSP is not larger than the minimal makespan detected for any other feasible mode-assignment.

Kolisch and Drexl [48] introduced a polynomial reduction from the knapsack problem, known to be $\mathcal{NP}$-complete [33], to the mode-assignment problem with $|N| \geq 2$ nonrenewable resources and $|M_j| \geq 2$, $\forall j \in J \backslash \{0, n+1\}$. Hence, with the latter input already the mode-assignment step is $\mathcal{NP}$-complete.

Now, to discuss the complexity of the single-mode scheduling step, let us consider the optimization problem (2.3), s.t. (2.4) and (2.6) with the variables (2.7) and the feasible mode-assignment $\bar{\mathbf{x}}$ at hand. W.l.o.g. we can omit all indices $\bar{x}_j \in M_j$ concerning the mode choice. As mentioned above, the resulting problem is the SRCPSP with GPRs, and in the special case $\delta_{i,j} = d_i$ the SRCPSP with SPRs. The problem $[PS, res^{\rho}1, 1, 1|d_j = 1, prec|D_{\max}]$ is a special case of the SRCPSP with SPRs with only one renewable resource with capacity one and where all jobs consume at most one unit of the resource and are of duration one. By a pseudopolynomial reduction [33] from the strongly $\mathcal{NP}$-hard 3-Partition-Problem, it follows that the latter problem ($[PS, res^{\rho}1, 1, 1|d_j = 1, prec|D_{\max}]$) is $\mathcal{NP}$-hard in the strong sense (See [33] and [13]).

The decision version of the latter problem, where a schedule for $[PS, res^{\rho}1, 1, 1|d_j = 1, prec]$ has to be found such that $D_{\max} \leq T$ for an arbitrary T is transformable to the special case $[PS, res^{\rho}1, 1, 1|d_j = 1, temp]$ of the feasibility problem of the SRCPSP with GPRs [10]. As this is a pseudopolynomial transformation [33] from a strongly $\mathcal{NP}$-hard problem, we can conclude that already the problem of finding a feasible schedule for the SRCPSP with GPRs is $\mathcal{NP}$-hard in the strong sense. In contrast, a feasible solution for the SRCPSP with SPRs can be found in polynomial time e.g. by the serial schedule generation scheme (SSGS) [6].

Another important complexity result was deduced by Schwindt [78]. He proposed a pseudopolynomial transformation from the partially ordered knapsack problem [33] to the problem of finding a feasible solution for the resource relaxation of the MRCPSP with GPRs, i.e. to the problem of finding mode-assignments $x_j \in M_j$ such that (2.4) is fulfilled. Hence, he showed that this problem is $\mathcal{NP}$-complete in the strong sense. Note, that the resource relaxation of the SRCPSP with SPRs and GPRs can be solved in polynomial time by longest path algorithms (See also Sect. 2.4).

2.4 Preprocessing techniques

In the following, we describe standard preprocessing techniques for the MRCPSP with GPRs from the literature which can also be found in [82] or [16]. These techniques can be divided into procedures eliminating modes and nonrenewable resources, and procedures deducing lower and upper bounds for the starting time variables s_i.

Firstly, we consider principles for the elimination of modes and nonrenewable resources. Note, that the different names for the eliminated parameters are the same as in [82].

In the first step, we can delete modes from the problem instance which are *non-executable* w.r.t. renewable and nonrenewable resources. A mode $m \in M_i$ is non-executable w.r.t. the renewable resource $r \in R$, if

$$c^{\rho}_{i,m,r} > C^{\rho}_{r}$$

and non-executable w.r.t. the nonrenewable resource $r \in N$, if

$$c^{\nu}_{i,m,r} + \sum_{j \in J \setminus \{i\}} \min\{c^{\nu}_{j,k,r} : k \in M_j\} > C^{\nu}_{r}.$$

Clearly, if there is one job with only non-executable modes, the respective problem instance is infeasible.

In the second step, we remove *redundant* nonrenewable resources. A nonrenewable resource is redundant, if

$$\sum_{j \in J} \max\{c^{\rho}_{j,k,r} : k \in M_j\} \leq C^{\nu}_{r}.$$

Note, that non-executable modes and redundant nonrenewable resources can be eliminated as this does not change the set of feasible solutions for the MRCPSP with GPRs.

The third step consists of eliminating *inefficient* modes. A mode $m \in M_i$ is defined as inefficient, if there exists another mode $m' \in M_i$ such that,

$$\begin{aligned} &d_{i,m'} \leq d_{i,m} \\ &c^{\rho}_{i,m',r} \leq c^{\rho}_{i,m,r}, \ \forall r \in R \\ &c^{\nu}_{i,m',r} \leq c^{\nu}_{i,m,r}, \ \forall r \in N \end{aligned}$$

If an optimal solution of an instance exists, the removal of inefficient modes does not lead to a change of the optimal makespan.

Note, that after the elimination of inefficient modes, we go back to the third step, as possibly new nonrenewable resources $r \in N$ can be removed.

The complete procedure is summarized in Algorithm 1.

After the application of Algorithm 1, we try to deduce new lower and upper bounds for the starting time variables s_i. Therefore we assume that an upper bound T for the project makespan is known. This can be the project makespan of a known feasible solution which was for example determined by a heuristic or $T_{\max}$ from (2.10).

With the above information at hand, the initial domains of the variables s_i, i.e. the lower bounds $lb(s_i)$ and upper bounds $ub(s_i)$ can be computed as follows (See also [16]).

Input: Mode sets $M_j,\ j \in J$ and nonrenewable resource set N
Output: New mode sets $M'_j \subseteq M_j,\ j \in J$ and nonrenewable resource set $N' \subseteq N$

$\forall j \in J,\ M'_j \leftarrow M_j;\ N' \leftarrow N;$
Remove non-executable modes w.r.t. renewable resources from M'_j for all $j \in J$;
Remove non-executable modes w.r.t. nonrenewable resources from M'_j for all $j \in J$;
if $\exists\, j \in J :\ M'_j = \emptyset$ **then**
  return *Instance infeasible*
end
repeat
  Remove redundant nonrenewable resources from N'.;
  Find the set B_j of inefficient modes in M'_j for all $j \in J$.;
  $\forall j \in J,\ M'_j \leftarrow M'_j \setminus B_j;$
until $(\forall j \in J\ :\ B_j = \emptyset) \vee N' = \emptyset;$
return $M'_j,\ j \in J$ *and* N'

Algorithm 1: Preprocessing: Elimination of modes and nonrenewable resources

In the presence of GPRs, we firstly determine the minimum timelag $\delta_{i,j}^{\min}$ for all precedence relations (i,j):

$$\delta_{i,j}^{\min} = \min_{k \in M_i, l \in M_j} \{\delta_{i,j,k,l}\} \tag{2.11}$$

Then, we generate a graph based on the precedence relations (i,j) with the arc weights $\delta_{i,j}^{\min}$.

We solve two longest path problems in the above graph with a procedure similar to the Bellman-Ford algorithm to determine the longest path $P(0,i)$ from the dummy node 0 to every node i and the longest path $P(i,n+1)$ from every node i to the dummy node $n+1$ unless the underlying MRCPSP instance is infeasible. If a negative cycle is found, the latter is the case. Otherwise, we can set $lb(s_i)$ to the length $l(P(0,i))$ of the longest path from 0 to i in the precedence network. $ub(s_i)$ can be set to $T - l(P(i,n+1))$.

As the dummy nodes 0 and $n+1$ represent the start and the end of the project (See Sect 2.1.3), it must hold that $l(P(0,i)) \geq 0$ and $l(P(i,n+1)) \geq \min_{k \in M_i}\{d_{i,k}\}$. Note that with SPRs we can use the more efficient forward-backward recursion procedure [16] to compute $lb(s_i)$ and $ub(s_i)$.

In the mathematical models for the MRCPSP with SPRs and GPRs in Chapt. 5 and Chapt. 6, we assume, that the parameters are those obtained after the application of the above preprocessing techniques.

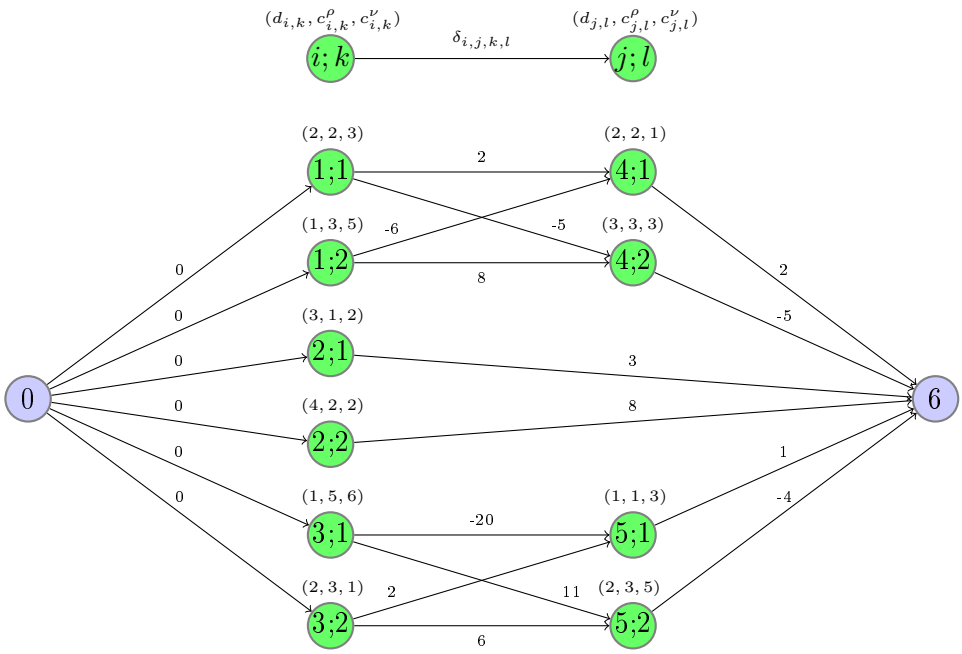

Figure 2.2: AoN network for example

2.5 An example for the MRCPSP with GPRs

Fig. 2.2 visualizes the precedence network for a small instance of the MRCPSP with GPRs with five jobs and two dummy jobs. For this example, we consider one renewable resource (machines) and one nonrenewable resource (a project budget) with maximal capacities $C^{\rho} = 3$ and $C^{\nu} = 10$, respectively.

The nodes correspond to all possible job-mode combinations and an arc is drawn between two nodes if there is a precedence relation between the jobs in the respective nodes[1].

Firstly, we apply the preprocessing procedures of the last section. Alg. 1 leads to the result, that the job-mode combinations 1;2, 3;1, 4;2 and 5;2 are non-executable w.r.t. the nonrenewable resources, i.e. these job-mode combinations cannot be included in a feasible schedule with a budget of 10 units. Moreover, the job-mode combination 2;2 is inefficient, i.e. the removal of 2;2 does not lead to an increase of the optimal makespan.

[1]Note, that in the general version of the MRCPSP with GPRs we would have to connect the dummy node 0 to every job $j \in J$ and every job $j \in J$ to the dummy job 6 to correctly model the start and end of the project. Because of the special structure in this example, this is not necessary and for the sake of simplicity, we only connect 0 to the jobs with no predecessors and the jobs with no successors to 6.

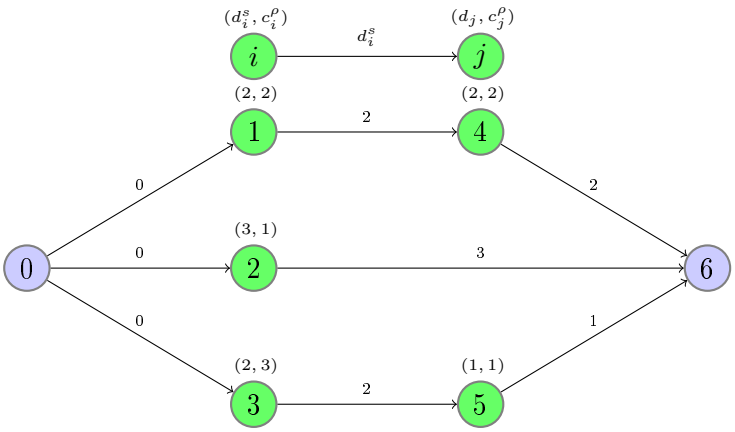

Figure 2.3: AoN network for the resulting SRCPSP

As now, every job can only be processed in one mode and the timelag on the arc leaving a job corresponds to the duration of the respective job, we are confronted with an instance of the SRCPSP with SPRs (see Fig. 2.3).

An ordered list $\mathcal{L}$ of jobs, which fulfills

$$\text{If } \mathcal{L}[m] \in \mathfrak{S}_{\mathcal{L}[n]} \Longrightarrow \; n < m$$

is called a precedence-feasible activity list (AL) [16]. The SSGS constructs feasible schedules for the SRCPSP with SPRs from precedence-feasible ALs [16].

The principle of the SSGS is simple: Iterate through $\mathcal{L}$ in ascending index order and schedule the respective activities at the earliest point in time such that the resulting schedule is resource and precedence feasible.

For the example of Fig. 2.3, we can determine 30 different precedence-feasible ALs. Let us for example consider the AL $\mathcal{L} = (0, 3, 1, 2, 4, 5, 6)$. If we apply the SSGS to $\mathcal{L}$ we obtain the feasible schedule visualized in Fig. 2.4 with the makespan $D_{\max} = 6$.

The makespan for our example project cannot be shorter than

$$LB_1 = \left\lceil \frac{\sum_{i \in J} c_{i,k}^\rho \cdot d_i}{C^\rho} \right\rceil$$

Thus, $LB_1 = 6$ is a lower bound on the makespan [16] and our schedule is by chance optimal.

Note, that in our example, we added two simplifications to the MRCPSP with GPRs. Firstly, there is only one nonrenewable resource, in which case finding a feasible mode assignment s.t. nonrenewable resource constraints is solvable in polynomial time. Secondly, the AoN network is acyclic. In this case, there is a polynomial time algorithm for the feasibility problem of the SRCPSP with GPRs [59] and of the resource relaxation of the MRCPSP with GPRs.

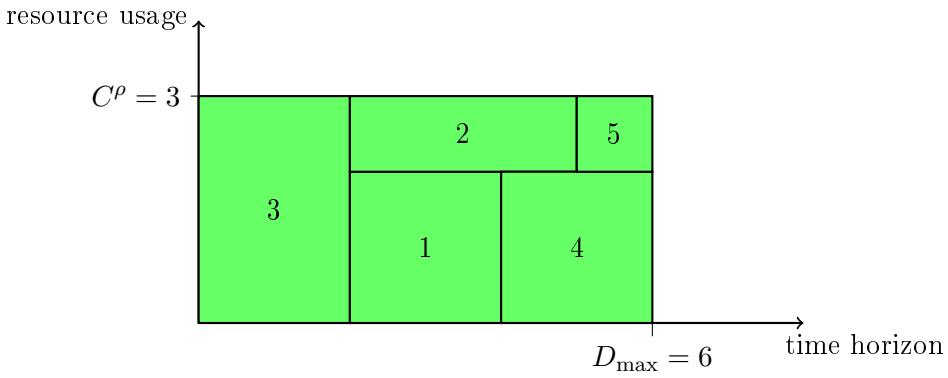

Figure 2.4: Visualisation of solution

Chapter 3

CP and SAT Solving for the RCPSP

3.1 Introduction to CP

The following section shall give a short overview over the theory of CP. It is mostly based on the books of Apt [5], Baptiste et al. [8] and Rossi et al. [66]. This section does not tackle the theory of CP in complete, i.e. the intent is to only provide the necessary basics to the reader.

3.1.1 Preliminaries

One can apply CP techniques to solve a constraint satisfaction problem (CSP) or a constraint optimization problem (COP).

A COP (CSP) is defined on a finite set of variables $X = \{x_1, \ldots, x_n\}$ with the domains $D_1, \ldots, D_n$, and on a set of constraints $\mathcal{C} = \{C_k(X_k) : k = 1 \ldots, p\}$ containing variables from the sets X_k with the index set I_k:

$$X_k = \{x_j : j \in I_k\} \subseteq X,\ k = 1, \ldots, p.$$

More precisely, a constraint $C_k(X_k)$ is a subset of the Cartesian product $\prod_{j \in I_k} D_j$.

Definition 3.1.1. *Let X be a finite set of variables with the domain set $\mathcal{D} = \{D_1, \ldots, D_n\}$. Moreover, let $\mathcal{C}$ be a finite set of constraints. With an objective function $o : D_1 \times \cdots \times D_n \mapsto \mathbb{R}$ to be minimized or maximized, we call the tuple $\mathcal{P} = (o, X, \mathcal{D}, \mathcal{C})$ a COP. If the problem $\mathcal{P}$ comes without an objective function, we call $\mathcal{P} = (X, \mathcal{D}, \mathcal{C})$ a CSP.*

In this dissertation, we assume w.l.o.g. that $\mathcal{P}$ is a minimization problem. Moreover, we restrict ourselves to finite integer domains $D_k = \{lb(x_k), \ldots, ub(x_k)\}$. Note, that one can also tackle domains consisting of real values with CP concepts [5].

A vector $\mathbf{a} = (a_1, \ldots, a_n)$ containing an assignment of the variables $x_k, k \in \{1, \ldots, n\}$ to the values $a_k \in D_k$ is called a feasible solution of the COP (CSP) $\mathcal{P}$, if

$(a_j)_{j \in I_k} \in C_k(X_k)$, $\forall k = 1, \ldots, p$. Moreover, if $o(\mathbf{a}) \leq o(\mathbf{a}')$ for all feasible solutions $\mathbf{a}'$, $\mathbf{a}$ is an optimal solution of the COP $\mathcal{P}$.

Note, that constraints can be stated explicitly by providing a concrete set of assignments $\mathcal{B}_k \subseteq \prod_{j \in I_k} D_j$ or implicitly by giving a formula on certain variables which has to be fulfilled, e.g. $(x_1)^3 + (x_2)^3 = (x_3)^3$, $x_i \in \{1, \ldots, n\}$, $\forall i = 1, 2, 3$.

In the context of CSPs, it is also important to define the notion of local and global consistency:

Definition 3.1.2. *Consistency*

1. *A CSP $\mathcal{P} = (X, \mathcal{D}, \mathcal{C})$ is **globally consistent**, if there exists a feasible solution to $\mathcal{P}$.*
2. *A CSP $\mathcal{P} = (X, \mathcal{D}, \mathcal{C})$ is **locally consistent** w.r.t. the subset $\mathcal{C}' \subseteq \mathcal{C}$, if $\mathcal{P}' = (X, \mathcal{D}, \mathcal{C}')$ is globally consistent.*
3. *The **constraint** $C_k(X_k)$ is called **locally consistent**, if $\mathcal{P} = (X, \mathcal{D}, \mathcal{C})$ is locally consistent w.r.t. the subset $\mathcal{C}' = \{C_k(X_k)\}$.*
4. *A CSP $\mathcal{P} = (X, \mathcal{D}, \mathcal{C})$ is **inconsistent**, if no feasible solution of $\mathcal{P}$ exists.*

CP solvers apply *constraint propagation* algorithms to decide about the consistency status of a CSP. A constraint propagation algorithm $prop : \mathcal{D} \rightarrow \mathcal{D}' = \{D'_1, \ldots, D'_n\}$ transforms $\mathcal{P} = (X, \mathcal{D}, \mathcal{C})$ into $\mathcal{P}' = (X, \mathcal{D}', \mathcal{C})$ such that

$$prop(D_k) = D'_k \subseteq D_k, \forall k = 1, \ldots, n.$$

Moreover, it holds that, if $\mathbf{a}$ is a feasible solution to $\mathcal{P}$, then $\mathbf{a}$ is a feasible solution to $\mathcal{P}'$. The aim is to obtain a problem $\mathcal{P}' = (X, \mathcal{D}', \mathcal{C})$, from which we can deduce a feasible solution of $\mathcal{P}$ or show that no such solution exists. Thus, constraint propagation algorithms try to prove global consistency or inconsistency of a CSP. Note, that if a constraint propagation algorithm obtains a problem $\mathcal{P}'$ such that there exists a $k \in \{1, \ldots, n\}$ with $D'_k = \emptyset$, we can deduce the inconsistency of the CSP $\mathcal{P}$.

Finding a feasible solution of a CSP is $\mathcal{NP}$-complete, as a CSP is a generalization of the SAT-problem [33]. Thus, it is unlikely that there is an efficient constraint propagation algorithm, which decides about global consistency or inconsistency of a CSP [1]. Therefore, in general constraint propagation algorithms aim at transforming a CSP into an equivalent version from which one can decide about the local consistency of the original CSP.

There are different notions of consistency w.r.t. constraints from which one can deduce local consistency. For example, constraint propagation algorithms can lead to *hyper-arc* consistency or *bound* consistency of a constraint.

Definition 3.1.3.

1. *A constraint $C_k(X_k)$ is **hyper-arc** consistent, if for all $j \in I_k$ and all $a_l^j \in D_j$, there exists a vector $\mathbf{a} = (a_i)_{i \in I_k} \in C_k(X_k)$ with $a_j = a_l^j$.*

2. *A constraint $C_k(X_k)$ is **bound** consistent, if for all $j \in I_k$ and all $a_l^j \in \{lb(x_j), ub(x_j)\}$, there exists a vector $\mathbf{a} = (a_i)_{i \in I_k} \in C_k(X_k)$ with $a_j = a_l^j$.*

Note, that hyper-arc consistency is a stronger condition than bound consistency, as $\{lb(x_j), ub(x_j)\} \subseteq D_j$.

Clearly, if a constraint is hyper-arc consistent or bound consistent, the CSP with only the latter constraint is globally consistent. However, hyper-arc consistency or bound consistency of every constraint of a CSP does not lead to global consistency of the latter. Note, that there exist numerous other consistency notions for constraints (see [5]).

Example 3.1.1. ***Constraint propagation to obtain hyper-arc (bound) consistency***
Consider a CSP with the variables $x_1 \in \{1,2,3,4\}$ and $x_2 \in \{1,2,3\}$ and the constraint

$$x_1 = d_{x_2}, \text{ where } \mathbf{d} = (d_1, d_2, d_3) = (3,4,5).$$

The above constraint is also known as the `element`-constraint in the CP community [85], as the variable x_1 is equal to $x_2 - th$ element of the vector $\mathbf{d}$.

A constraint propagation algorithm which firstly updates the domain of x_1 to $D_1 \cap \{d_{x_2} : x_2 \in \{1,2,3\}\}$, i.e. to $\{3,4\}$ and then D_2 to $\{x_2 : x_2 \in \{1,2,3\} \wedge d_{x_2} \in \{3,4\}\}$, i.e. to $\{1,2\}$, leads to hyper-arc consistency (and clearly also bound consistency) of the above constraint.

As mentioned above, it is unlikely to efficiently decide about global consistency or inconsistency of a CSP solely by constraint propagation. Therefore, CP solvers apply a combination of search and constraint propagation. Backtracking search in combination with constraint propagation is a well-known algorithm to solve a CSP (see Algorithm 2).

The idea of backtracking with constraint propagation (BCP) is straightforward. In one iteration of the **repeat**-loop starting in line 3, the first element is taken from a list of nodes L at which a node corresponds to a subproblem $\mathcal{P}^{\text{act}}$ of $\mathcal{P}$. Firstly, a constraint propagation algorithm *prop* updates the variable domains in $\mathcal{D}^{\text{act}}$. If there exists one variable with empty domain in $\mathcal{D}^{\text{act}}$ after propagation, we continue with the next iteration of the **repeat**-loop or leave the **repeat**-loop. If the domains of all variables only consist of one variable, the respective solution $\mathbf{a}$ is tested. If $\mathbf{a}$ is feasible, $\mathbf{a}$ is returned and the algorithm is aborted. Otherwise, we proceed with the next iteration of the **repeat**-loop. If there exists at least one variable with at least two values in D_i^{act}, we choose one of these variable x_i and split the integer domain w.r.t. the value $v_i \in D_i^{\text{act}}$. In the next iteration of the **repeat**-loop, the

Input: CSP $\mathcal{P} = (X, \mathcal{D}, \mathcal{C})$.
Output: Feasible solution **a** or `Instance infeasible` (no solution exists).

$L \leftarrow \emptyset$;
Append $\mathcal{P}$ to L;
repeat
  $\mathcal{P}^{\text{act}} \leftarrow L[0]$;
  Remove $L[0]$ from L;
  $\mathcal{D}^{\text{act}} \leftarrow prop(\mathcal{D}^{\text{act}})$;
  if $\exists\, x_i \in X :\ D_i^{\text{act}} = \emptyset$ **then**
    `continue`;
  end
  if $\forall\, x_i \in X :\ D_i^{\text{act}} = \{a_i\}$ **then**
    if $\mathbf{a} = (a_1, \ldots, a_n)$ *is a feasible solution.* **then**
      return a;
    else
      `continue`;
    end
  end
  Choose unassigned variable x_i in $\mathcal{P}^{\text{act}} = (X, \mathcal{D}^{act}, \mathcal{C})$ and value $v_i \in D_i^{act}$;
  $D_i^{act,1} \leftarrow \{lb(D_i^{act}), \ldots, v_i\}$; $D_i^{act,2} \leftarrow \{v_i + 1, \ldots, ub(D_i^{act})\}$;
  Prepend $\mathcal{P}^2 = (X, \mathcal{D}^{act,2}, \mathcal{C})$ to L ; Prepend $\mathcal{P}^1 = (X, \mathcal{D}^{act,1}, \mathcal{C})$ to L;
until $L = \emptyset$;
return *Instance infeasible*

Algorithm 2: Backtracking with constraint propagation

problem D_i^{act} with the new domain $D_i') = \{lb(D_i^{act}), \ldots, v_i\}$ is considered, as $\mathcal{P}_i^{act,1}$ is prepended to L after $\mathcal{P}_i^{act,2}$ (see line 19 in Algorithm 2). Algorithm 2 terminates by returning a feasible solution or by returning that the instance $\mathcal{P}$ is infeasible.

In our presentation of BCP, we choose a *branching* strategy known as *domain splitting* [66] (see line 17 in Algorithm 2). Other strategies for creating new choice points in the search tree are *2-way* or *d-way* branching [66]. 2-way branching creates two new nodes from the incumbent node by selecting a value v_i from the domain of the chosen variable x_i and adding the constraint $x_i = v_i$ to P^{act} in the first node and the constraint $x_i \neq v_i$ to the second node. *d-way* branching (also known as enumeration) creates $|D_i^{\text{act}}|$ new choice points by successively adding the constraints $x_i = v_i$ for all $v_i \in D_i^{\text{act}}$ to P^{act}. Hwang and Mitchell [45] show that 2-way branching is strictly more powerful than d-way branching. It can be easily seen that d-way branching can be imitated by 2-way branching without a significant loss of efficiency. To prove their claim, the authors show that the claim exchanging 2-way with d-way in the last statement is not true. In addition, Van Hentenryck [84] compares the d-way branching strategy to the domain splitting strategy.

To decide about the selection of the unassigned variable and a corresponding domain value, CP solvers apply *variable* and *value ordering* heuristics. Different variants for ordering the variables and respective domain values are discussed in [66]. One example for a technique for variable and value selection is to choose the unassigned variable with the smallest domain and then the first value of the associated variable domain. In Sect. 3.2 and Sect. 3.3, we discuss a more sophisticated variable and value ordering heuristics which has turned out to be efficient in SAT solving and also in the context of the RCPSP.

BCP explores the search tree with a depth-first search strategy. This is due to the fact, that in the **repeat**-loop, the algorithm always selects a node from the currently maximal depth level in the search tree based on the variable and value ordering heuristic.

Limited discrepancy search is another technique for the search tree exploration which differs from the depth-first search principle [38]. With this technique, it can be the case that a node from a depth level m can be processed before a node from the depth level n, where $m < n$. Harvey and Ginsberg [38] theoretically and experimentally show, that limited discrepancy search is a more effective way to explore the search tree in the context of globally consistent CSPs compared to the depth-first search strategy used in BCP.

BCP leads to a *conflict* or *dead end*, if both choice points arising from a node lead to inconsistent CSPs, i.e. if one of the conditions in lines 7 and 13 of Algorithm 2 is true for both choice points. In case of a conflict, Algorithm 2 backtracks from at node a the depth level m to a node at the depth level $m-1$. Hence, it backtracks *chronologically*. In some cases, it can be possible to backtrack from the depth level m to the depth level n, where $n < m-1$, i.e. to backtrack *non-chronologically*. This technique is also known as *backjumping* [66]. We will discuss backjumping in the

context of SAT solving in Sect. 3.2 and in the context of an exact solution method for the RCPSP in Sect. 3.3.

BaB algorithms with constraint propagation can be applied to find an optimal solution of a COP (see Algorithm 3). The principle of BaB with constraint propagation is similar to the BaB algorithm implemented in MIP solvers introduced by Dakin [21].

The node with the highest priority is selected from the list L (see line 6 in Algorithm 3). Achterberg [2] summarizes and evaluates different strategies for calculating the priority of a node in the context of MIP. These are based on the objective value of the linear programming relaxation of the MIP formulation stored in a node. His proposed node selection strategies can be generalized for the application in a BaB algorithm with constraint propagation.

Therefore, a lower bounding function $bound^o : 2^{D_1 \times \cdots \times D_n} \to \mathbb{R}$ depending on the objective function o has to be given, where $2^{D_1 \times \cdots \times D_n}$ represents the power set of $D_1 \times \cdots \times D_n$. $bound^o$ has to fulfill the following two conditions:

$$F_1 \subseteq F_2 \Rightarrow bound^o(F_1) \geq bound^o(F_2) \tag{3.1}$$

$$bound^o(F_1) \leq o(\mathbf{x}), \forall x \in F_1 \tag{3.2}$$

$bound^o$ is also applied to decide if a node can be pruned (see line 21 in Algorithm 3).

Moreover, every time a better solution is found, we add a constraint to the constraint set $\mathcal{C}$, forcing the objective value of new solutions to be better than or equal to the best known value $o(\mathbf{a}^*)$ (see line 15 in Algorithm 3). Note, that the underlying CP solver has to provide a mechanism for the handling of the added constraint on the objective function.

In many applications the objective function can be modeled as a linear combination of the decision variables $o(\mathbf{x}) = \mu_1 \cdot x_1 + \cdots + \mu_n \cdot x_n$. In this case, we can define $bound^o$ as follows:

$$bound^o(D_1^{act} \times \cdots \times D_n^{act}) = \mu_1 \cdot x_1^{lb} + \cdots + \mu_n \cdot x_n^{lb},$$
$$\text{where } x_k^{lb} = \begin{cases} lb(D_k^{act}), & \text{if } \mu_k \geq 0 \\ ub(D_k^{act}), & \text{else} \end{cases}$$

This is a correct lower bounding function for the above objective function, as it fulfills (3.1) and (3.2).

Moreover in this case, the constraint $\mu_1 \cdot x_1 + \cdots + \mu_n \cdot x_n \leq o(\mathbf{a}^*)$ can be added to the constraint set. A constraint propagation algorithm on the above constraint can deduce the following upper bound update:

$$ub(D_k^{act}) \leftarrow \begin{cases} \min\left\{ub(D_k^{act}), \left\lfloor \frac{a^* - \sum_{i \neq k} \mu_j \cdot x_j^{lb}}{\mu_k} \right\rfloor\right\}, & \text{if } \mu_k \geq 0 \\ ub(D_k^{act}), & \text{else} \end{cases} \tag{3.3}$$

Input: COP $\mathcal{P} = (o, X, \mathcal{D}, \mathcal{C})$.
Output: Optimal solution $\mathbf{a}^*$ or `Instance infeasible` (no solution exists).

$L \leftarrow \emptyset$;
$o(\mathbf{a}^*) \leftarrow \infty$;
Calculate priority *prio* of node $\mathcal{P}$;
Append $(prio, \mathcal{P})$ to L `// Elements of` L `sorted by descending` *prio* `of nodes`;
repeat
  $\mathcal{P}^{\text{act}} \leftarrow (L.first).value$;
  Remove $L.first$ from L;
  $\mathcal{D}^{\text{act}} \leftarrow prop(\mathcal{D}^{\text{act}})$;
  if $\exists\, x_i \in X :\ D_i^{\text{act}} = \emptyset$ **then**
    `continue`;
  end
  if $\forall\, x_i \in X :\ D_i^{\text{act}} = \{a_i\}$ **then**
    if $(\mathbf{a} = (a_1, \ldots, a_n)$ *is a feasible solution*$) \wedge (o(\mathbf{a}) < o(\mathbf{a}^*))$ **then**
      $\mathbf{a}^* \leftarrow \mathbf{a}$;
      $\mathcal{C} \leftarrow \mathcal{C} \cup \{o(\mathbf{x}) \leq o(\mathbf{a}^*)\}$;
      `continue`;
    else
      `continue`;
    end
  end
  if $bound^o(D_1^{act} \times \cdots \times D_n^{act}) \geq o(\mathbf{a}^*)$ **then**
    `continue`;
  end
  Choose unassigned variable x_i in $\mathcal{P}^{\text{act}} = (X, \mathcal{D}^{act}, \mathcal{C})$ and value $v_i \in D_i^{act}$;
  $D_i^{act,1} \leftarrow \{lb(D_i^{act}), \ldots, v_i\}$; $D_i^{act,2} \leftarrow \{v_i + 1, \ldots, ub(D_i^{act})\}$;
  Calculate $prio_1$ and $prio_2$ for $\mathcal{P}^1 = (X, \mathcal{D}^{act,1}, \mathcal{C})$ and $\mathcal{P}^2 = (X, \mathcal{D}^{act,2}, \mathcal{C})$;
  Add $(prio_1, \mathcal{P}^1)$ and $(prio_2, \mathcal{P}^2)$ to L;
until $L = \emptyset$;
if $o(\mathbf{a}^*) = \infty$ **then**
  return *Instance infeasible*
else
  return $\mathbf{a}^*$
end

Algorithm 3: BaB with constraint propagation

Moreover, it can deduce the following lower bound update:

$$lb(D_k^{act}) \leftarrow \begin{cases} \max\left\{lb(D_k^{act}), \left\lfloor \frac{a^* - \sum_{i \neq k} \mu_j \cdot x_j^{ub}}{\mu_k} \right\rfloor\right\}, & \text{if } \mu_k < 0 \\ lb(D_k^{act}), & \text{else} \end{cases} \tag{3.4}$$

$$\text{where } x_k^{ub} = \begin{cases} ub(D_k^{act}), & \text{if } \mu_k \geq 0 \\ lb(D_k^{act}), & \text{else} \end{cases}$$

If the domains of all variables $x_k \in X$ are updated based on (3.3) and (3.4), this leads to bound consistency of the constraint on the objective function or to inconsistency of the latter.

Achterberg [1] describes an efficient procedure for updating the domains based on (3.3) and (3.4) in the course of the BaB algorithm without having to recalculate the complete term on the right hand side of (3.3) and (3.4) in every node of the BaB tree and to only consider situations, where there is a possibility of an update. He also proposes strategies to avoid numerical errors, which can appear if the coefficients μ_k are real numbers.

3.1.2 CP for scheduling problems

Standard CP solvers offer the possibility to model complicated problem specific structures by *global constraints* [5]. Informally speaking, a problem structure which is normally expressed by a number of e.g. linear constraints is captured in one expression. Bessière and Van Hentenryck [12] provide a formal definition of global constraints.

`cumulative` is a well-known global constraint for the modeling of renewable resources in the context of scheduling problems [4]. In case of a SRCPSP, we can model the renewable resource constraints (2.6) by the following linear constraints [50]:

$$\sum_{t \in \{lb(s_i), \ldots, ub(s_i)\}} y_{i,t} = 1, \quad i \in J \tag{3.5}$$

$$\sum_{i \in J} \sum_{s=\max\{t-d_i+1\,;\,lb(s_i)\}}^{\min\{t\,;\,ub(s_i)\}} c_{i,r}^{\rho} \cdot y_{i,t} \leq C_r^{\rho}, \ \forall t = 0, \ldots, T-1, \ \forall r \in R \tag{3.6}$$

$$y_{i,t} \in \begin{cases} \{0,1\}, & \text{if } t \in \{lb(s_i), \ldots, ub(s_i)\} \\ \{0\}, & \text{if } t \in \{0, \ldots, T\} \setminus \{lb(s_i), \ldots, ub(s_i)\} \end{cases}, \forall i \in J \tag{3.7}$$

With the binary decision variables $y_{i,t}, \ t = 0, \ldots, T$, the starting time of job i is modeled, i.e. $y_{i,t}$ is equal to one if job i starts at time t and zero else.

CP solvers like e.g. Gecode [34] provide the possibility to model renewable resource constraints (2.6) in a more compact way through `cumulative`:

$$\texttt{cumulative}(\mathbf{s}, \mathbf{d}, \mathbf{c}_r^{\rho}, C_r^{\rho}), \quad \forall r \in R$$

In the above formulation, the usage of $|n+2| \cdot (ub(s_i) - lb(s_i) + 1)$ binary variables is omitted, i.e. we can directly use the integer variables $s_i \in \{lb(s_i), \ldots, ub(s_i)\}$. Furthermore, there are only $|R|$ `cumulative` constraints to model renewable resources, whereas (3.6) integrates $|R| \cdot T$ linear constraints.

Besides a feasibility check w.r.t. (2.6) (without the index x_j in case of the SR-CPSP), `cumulative` integrates specialized constraint propagation algorithms like e.g. *TP* and *edge finding (EF)* [8]. These algorithms lead to stronger domain reductions compared to propagating every linear constraint from (3.5) - (3.6) separately.

If the deadline T is arbitrary, the problem of finding a feasible schedule w.r.t. the renewable resource constraints with $s_{n+1} \leq T$ is already $\mathcal{NP}$-complete for the single mode case with $|R| = 1$ [73]. Thus, it is unlikely that there exists an efficient constraint propagation algorithm which obtains hyper-arc or bound consistency of `cumulative`. TP and EF are all polynomial constraint propagation algorithms and maintain a weaker level of local consistency.

TP works with *compulsory parts* cp_j of the jobs $j \in J$ [74]:

$$cp_j = \begin{cases} \{ub(s_j), \ldots, lb(s_j) + d_j - 1\}, & \text{if } lb(s_j) + d_j > ub(s_j) \\ \emptyset, & \text{else} \end{cases} \tag{3.8}$$

If $cp_j \neq \emptyset$, we can conclude that job j is surely processed in every time interval $[t, t+1[$, where $t \in cp_j$.

On the one hand, TP can detect an inconsistency, i.e. a time point t_κ at which there is a violation of the renewable resource capacity of $r \in R$:

$$\sum_{j:t_\kappa \in cp_j} c^\rho_{j,r} > C^\rho_r \tag{3.9}$$

An efficient algorithm for the inconsistency check can be implemented with a time complexity of $O(|J| \cdot \log(|J|))$ with data structures called *resource profiles*[1] [73].

If no inconsistency is detected, TP tries to deduce lower and upper bound updates. If for job $i \in J$ there is a time point $lb(s_i) \leq t^* \leq lb(s_i) + d_i - 1$ such that

$$c^\rho_{i,r} + \sum_{j \neq i: t^* \in cp_j} c^\rho_{j,r} > C^\rho_r, \tag{3.10}$$

it follows, that s_i must be greater than t^*, i.e. we can update $lb(s_i) \leftarrow t^* + 1$.

This procedure can be repeated until there is a time point $lb(s_i) \leq t^* = t_{\max} \leq lb(s_i) + d_i - 1$ which either fulfills (3.10) and

$$\forall t: \; t_{\max} + 1 \leq t \leq t_{\max} + d_i \; : \qquad c^\rho_{i,r} + \sum_{j \neq i: t \in cp_j} c^\rho_{j,r} \leq C^\rho_r \tag{3.11}$$

$$\text{and} \quad t_{\max} < ub(s_i) \tag{3.12}$$

[1] These are also known as *eventlists* [6].

or (3.10) and $t_{\max} \geq ub(s_i)$. In the first case, $t_{\max}+1$ is the best possible lower bound for job i deducible by TP. In the second case, TP again found an inconsistency. Note, that TP applies a symmetric procedure to process upper bound updates.

An efficient TP algorithm for the deduction of possible domain updates for all jobs $j \in J$, can be implemented with a time complexity of $O(|J|^2)$ again using resource profiles [73].

EF applies reasoning based on the *energy* $e_i = d_i \cdot c^{\rho}_{i,r}$ of the jobs $i \in J$ w.r.t. the renewable resource $r \in R$. Therefore, let $\Psi \subseteq J$ be a set of jobs and let $lb(\Psi)$, $ub(\Psi)$ and e_Ψ be defined as follows:

$$lb(\Psi) = \min_{i \in \Psi}\{lb(s_i)\}$$
$$ub(\Psi) = \max_{i \in \Psi}\{ub(s_i) + d_i\}$$
$$e_\Psi = \sum_{i \in \Psi} e_i$$

A SRCPSP instance is *energy-feasible* w.r.t. resource $r \in R$, if:

$$\forall\ \Psi \subseteq J:\ e_\Psi \leq (ub(\Psi) - lb(\Psi)) \cdot C^{\rho}_r.$$

Note, that if a SRCPSP instance is resource-feasible w.r.t. $r \in R$, i.e. it fulfills (2.6), it is also energy-feasible but the converse does not hold.

Hence, the following theorem can be stated [8]:

Theorem 3.1.1. *If there exists a job $i \in J$ and a set $\Psi \subseteq J\backslash\{i\}$, such that:*

$$(ub(\Psi) - lb(\Psi \cup \{i\})) \cdot C^{\rho}_r < e_{\Psi \cup \{i\}}, \tag{3.13}$$

then all jobs $j \in \Psi$ have to end before the end of job i.

Assume now, that we found a job i and a set Ψ fulfilling the above theorem. Moreover, let $rest(\Psi, i)$ be defined as follows:

$$rest(\Psi, i) = e_\Psi - (C^{\rho}_r - c^{\rho}_{i,r}) \cdot (ub(\Psi) - lb(\Psi)).$$

Informally speaking, $rest(\Psi, i)$ is the energy consumed by the jobs $j \in \Psi$ which is above or below the rectangle with the area $(C^{\rho}_r - c^{\rho}_{i,r}) \cdot (ub(\Psi) - lb(\Psi))$.

If $rest(\Psi, i) > 0$, there are at least $\lceil rest(\Psi, i)/c^{\rho}_{i,r} \rceil$ time points in the interval $[lb(\Psi), ub(\Psi)[$ at which job i cannot be processed. One can see, that in the above case job i cannot start at any time point $t \in \{lb(s_i), \ldots, lb(\Psi) + \lceil rest(\Psi, i)/c^{\rho}_{i,r} \rceil - 1\}$ as this would lead to a violation of Theorem 3.1.1 or a resource overload. Hence, $lb(s_i)$ can be updated to $lb(\Psi) + \lceil rest(\Psi, i)/c^{\rho}_{i,r} \rceil$.

If all jobs $j \in \Psi$ must end before the end of i, then also all jobs $j \in \Psi' \subseteq \Psi$ must end before the end of i. Moreover, there can be different sets $\Psi \subseteq J\backslash\{i\}$, fulfilling

(3.13). An EF algorithm deduces the maximal possible lower bound update for every job $i \in J$, i.e. it computes:

$$lb^{\max}(s_i) = \max_{\Psi \subseteq J \setminus \{i\}: \Psi \text{ fulfills (3.13)}} \max_{\Psi' \subseteq \Psi: rest(\Psi', i) > 0} \{lb(\Psi') + \lceil rest(\Psi', i)/c^{\rho}_{i,r} \rceil\}. \quad (3.14)$$

Note that similar updates can be deduced for $ub(s_i)$, if we "invert" the instance, i.e. maximal ending times $ub(s_i)+d_i$ are converted to minimal starting times $lb'(s_i) = T-(ub(s_i)+d_i)$ and minimal starting times $lb(s_i)$ to maximal ending times $ub'(s_i)+d_i = T - lb(s_i)$.

Baptiste et al. [8] propose an EF algorithm with a worst case time complexity $O(|J|^2)$. Mercier and Van Hentenryck [55] show, that this algorithm is incomplete, i.e. it does not compute the maximal possible lower bound w.r.t. (3.14). They also provide a complete EF algorithm running in $O(|J|^2 \cdot k)$ where $k \leq |J|$ is the number of different values in $\{c^{\rho}_{1,r}, \ldots, c^{\rho}_{n,r}\}$ for $r \in R$. Vilím [88] improves the worst case time complexity to $O(|J| \cdot log(|J|) \cdot k)$.

Baptiste et al. [8] theoretically compare many different propagation algorithms for `cumulative`. Nothing can be said about TP being stronger than EF or vice versa. However, they show that both TP and EF are weaker than *energetic reasoning* [8, p.67] which has a worst case time complexity of $O(|J|^3)$.

The following two examples show possible deductions based on EF and TP, and that neither of the two are better than the other one.

Example 3.1.2. ***EF is not stronger than TP***

One renewable resource with a maximal capacity $C^{\rho} = 2$ and two jobs with the following input are given:

$$(domain(s_1), d_1, c^{\rho}_1) = (\{3,4,5\}, 2, 1)$$
$$(domain(s_2), d_2, c^{\rho}_2) = (\{2,3,4\}, 3, 2)$$

The compulsory part of job 2 $cp_2 = \{4\}$. As $lb(s_1) + d_1 = 3 + 2 \geq 4$, starting job 1 at its lower bound would lead to a resource conflict at the time point $t_{max} = 4$. As t_{max} fulfills (3.10), (3.11) and (3.12), $t_{max} + 1 = 5$ is the maximal possible lower bound for job 1 deducible by TP. The respective lower bound update is visualized in Fig. 3.1.

EF does not lead to a lower bound update of job 1, as

$$(ub(\{2\}) - lb(\{1,2\})) \cdot 2 \geq 2 \cdot 1 + 3 \cdot 2$$

Hence, both jobs 1 and 2 fit in the respective rectangle (see Fig. 3.2). Thus, there is no set Ψ for job 1 fulfilling the condition of Theorem 3.1.1, which is necessary for a lower bound update of job 1 based on EF.

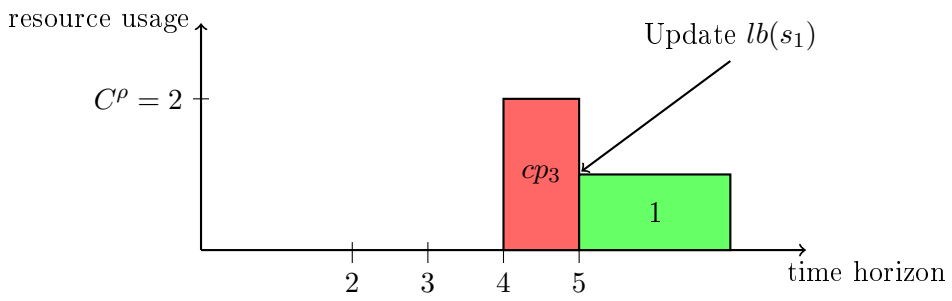

Figure 3.1: Example 3.1.2, TP

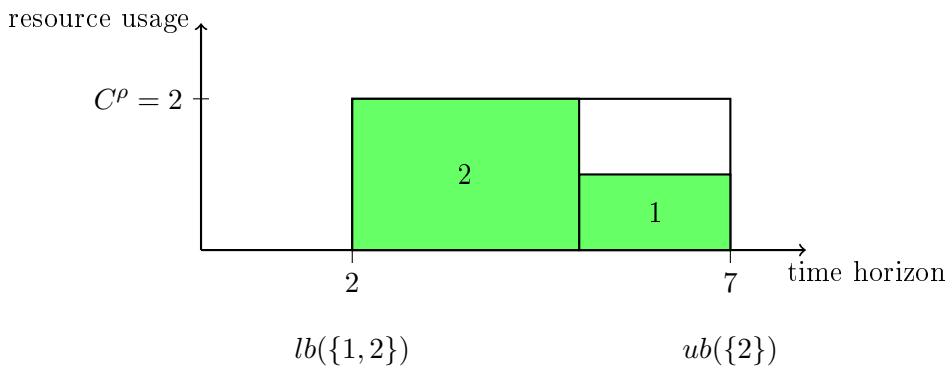

Figure 3.2: Example 3.1.2, EF, Theorem 3.1.1 not applicable for job 1

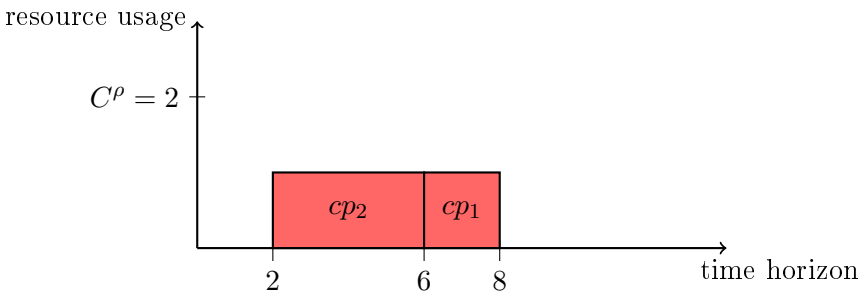

Figure 3.3: Example 3.1.3: TP, compulsory parts

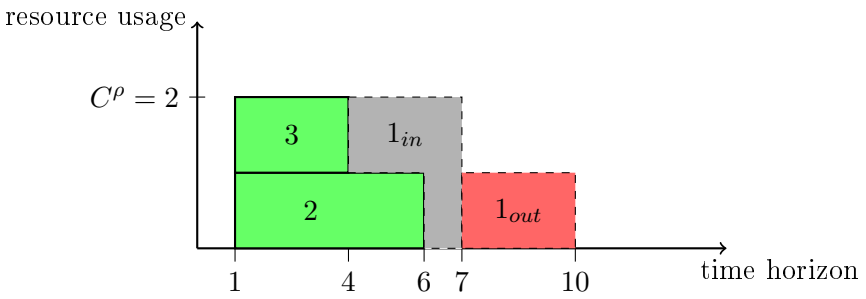

Figure 3.4: Example 3.1.3, EF, Theorem 3.1.1 applies for job 1

Example 3.1.3. ***TP is not stronger than EF***
One renewable resource with the maximal capacity $C^\rho = 2$ and three jobs with the following input are given:

$$(domain(s_1), d_1, c_1^\rho) = (\{1, \ldots, 6\}, 7, 1)$$
$$(domain(s_2), d_2, c_2^\rho) = (\{1, 2\}, 5, 1)$$
$$(domain(s_3), d_3, c_3^\rho) = (\{1, \ldots, 4\}, 3, 1)$$

Job 1, job 2 and job 3 have the compulsory parts $cp_1 = \{6, 7\}$, $cp_2 = \{2, \ldots, 5\}$ and $cp_3 = \emptyset$, respectively (see Fig. 3.3). TP does not lead to a lower bound update of job 1, as it can be scheduled at its lower bound without causing a resource violation.

EF can possibly deduce a lower bound update, as condition (3.13) of Theorem 3.1.1 is fulfilled for job 1 and $\Psi = \{2, 3\}$ (see also Fig. 3.4):

$$(ub(\{2, 3\}) - lb(\{1, 2, 3\})) \cdot C^\rho = 12 < 15 = \sum_{i=1}^{3} c_i^\rho \cdot d_i.$$

As $rest(\{2, 3\}, 1) = 2 > 0$, it holds, that the starting time of job 1 must be greater than or equal to three (see Fig. 3.5). One can verify that this is the maximal possible lower bound for job 1 which can be deduced by EF, i.e. $lb^{\max}(s_1) = 3$.

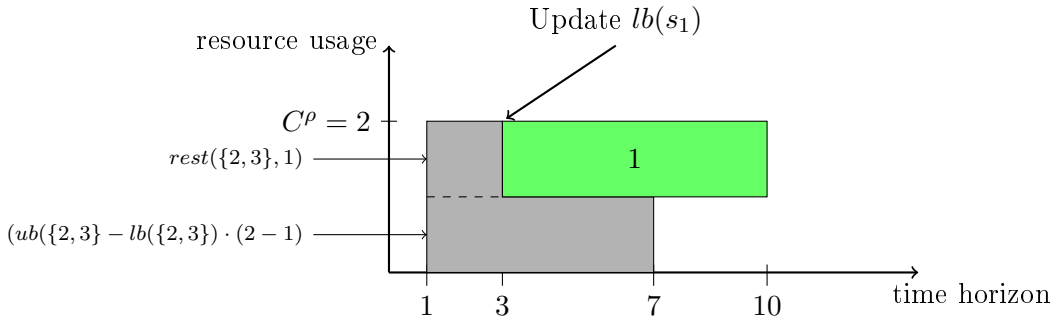

Figure 3.5: Example 3.1.3, EF, Lower bound update based on (3.14)

Vilím [89] proposes an algorithm combining TP and EF which is stronger than EF but still weaker than energetic reasoning. TP, EF and the combination of both are the most widely used constraint propagation rules for `cumulative`, as they have a lower worst case time complexity compared to energetic reasoning.

3.2 Introduction to SAT Solving

The SAT problem can be defined as follows:

Definition 3.2.1. *Find an assignment for the boolean variables $x_i \in \{0,1\}, i = 1,\ldots,n$, such that a given conjunction of clauses (disjunctions) consisting of the literals x_i or $\neg x_i$ is true or prove that no such assignment exists.*

The conjunction $\mathfrak{B}$ under consideration can be written as follows:

$$\begin{aligned} &\mathfrak{B} = \mathfrak{D}_1 \wedge \cdots \wedge \mathfrak{D}_p, \\ \text{where } &\mathfrak{D}_k = l_1^k \vee \cdots \vee l_{i_k}^k, \ \forall k = 1,\ldots,p, \\ &\exists\, i \in \{1,\ldots,n\} \ : \ l_j^k \in \{x_i, \neg x_i\}, \ \forall j = 1,\ldots,i_k, \ \forall k = 1,\ldots,p, \\ &x_i \in \{0,1\}, i = 1,\ldots,n. \end{aligned}$$

The SAT-problem was the first problem proven to be $\mathcal{NP}$-complete [33]. By a polynomial transformation of the SAT problem to the 3-SAT-problem, the SAT problem consisting of clauses with up to three literals, it can be shown that even this rather small SAT problem is already $\mathcal{NP}$-complete [33].

Davis and Putnam [22] were the first to propose a solution approach for the SAT problem. The refined version of the latter approach proposed by Davis, Logemann, and Loveland [23] is known as the Davis–Putnam–Logemann–Loveland (DPLL) algorithm. The DPLL algorithm is a backtracking algorithm similar to Algorithm 2 in the context of CP solving. In every iteration, it creates two new choice points

by choosing an unassigned variable x_i and assigning it to 0 (`false`) and 1 (`true`) successively (see line 17 in Algorithm 2).

The propagator *prop*(...) (see line 6 in Algorithm 2) applies unit propagation (UP). UP processes the following update for the clauses $\mathfrak{D}_k,\ k = 1, \ldots, p$:

$$(\exists\, j \in \{1, \ldots, i_k\} : \bigvee_{t \neq j} l_t^k = 0) \Longrightarrow l_j^k = 1 \tag{3.15}$$

An efficient implementation of UP can be realized by the integration of the *two watched literal* scheme [56]. An advantage of this procedure is that clauses do not have to be checked for every assignment of its literals but only when one of the two watched literals is assigned to zero. Another advantage is that the watched literals do not have to be changed when the DPLL algorithm backtracks [1]. The experiments of Moskewicz et al. [56] show that the integration of the two watched literal scheme for the implementation of UP can lead to significant performance improvements compared to other state-of-the-art implementations at that time.

The DPLL algorithm additionally applies *pure literal elimination*. If there exists a boolean variable x_i such that there are no pairs of literals $l_{j_1}^{k_1}, l_{j_2}^{k_2} \in \{x_i, \neg x_i\},\ k_1 \neq k_2$ with $l_{j_1}^{k_1} = \neg l_{j_2}^{k_2}$, the clauses containing literals $l \in \{x_i, \neg x_i\}$ can be eliminated. This is due to the fact, that we can always find an assignment for the respective variable x_i such that the clauses containing x_i are true.

Modern SAT solvers are based on the DPLL algorithm in combination with conflict analysis (CA) techniques (see [53] and [91]). A conflict occurs if through UP a clause $\mathfrak{D}_s$ becomes false, i.e. $l_1^s \vee \cdots \vee l_{i_k}^s = 0$.

The CA mechanism tries to find new clauses for the SAT problem based on the conflict at hand. These new clauses can possibly prune nodes of the search tree in later iterations. Furthermore, CA can deduce backjumps, i.e. with CA, the DPLL algorithm becomes a non-chronological backtracking procedure. The principles of CA are outlined in Sect. 3.2.1. Our description of CA is mainly based on the publications of Marques-Silva and Sakallah [53], Zhang et al. [91], Achterberg [1] and Schutt [73].

Another important part of the DPLL algorithm is the choice of the boolean variable and respective value based on which the search continues. Modern SAT solvers apply a conflict driven variable and value selection strategy [56]. Moreover, periodical restarts of the SAT solver with information from the previous run have turned out to be efficient [56]. We discuss the concepts of conflict driven search and restarts in Sect. 3.2.2.

There exist many different implementations of SAT solvers, which differ in the CA procedures and the search strategies. Zhang et al. [91] experimentally evaluate various implementations.

3.2.1 Conflict Analysis

Every time the DPLL algorithm detects a conflict, the CA mechanism is initialized by building up a *conflict graph* $G = (N, E)$. The nodes $N \subseteq \{x_1, \ldots, x_n, \neg x_1, \ldots, \neg x_n\} \cup$

$\{\kappa\}$ of the directed graph G correspond to the literals involved in the conflict and the conflict κ itself. An arc $e = (\neg l_s^k, l_j^k) \in E$ is drawn between the literals $\neg l_s^k$ and l_j^k, if UP applied to the clause $\mathfrak{D}_k$ led to the deduction $l_j^k = 1$ and

$$(\bigvee_{t \neq s,j} l_t^k) \vee l_s^k = 0$$

was the reason for the deduction (see 3.15). Moreover, an arc $e = (\neg l_j^\kappa, \kappa) \in E$ is drawn from $\neg l_j^\kappa$ to the conflict node κ, if l_j^κ is an element of the conflict clause $\mathfrak{D}_\kappa$, i.e.

$$(\bigvee_{t \neq j} l_t^\kappa) \vee l_j^\kappa = 0.$$

Moreover, we distinguish between branching nodes (literals), i.e. nodes which were assigned because of a branching decision and deduction nodes, i.e. nodes which were assigned because of a deduction based on UP.

Every node also integrates information about the *decision level*, i.e. the depth level in the search tree, where the respective literal was assigned.

For the deduction of a conflict clause the node set is partitioned into two sets N_r and N_c with $N_r \cup N_c = N$ and $N_r \cap N_c = \emptyset$. N_r contains all branching nodes and N_c contains the conflict node. We can deduce the following conflict clause $\mathfrak{D}$ for the problem at hand from the set $K = \{l_r \in N_r \ : \ \exists\, l_c \in N_c \text{ with } (l_r, l_c) \in E\}$:

$$\mathfrak{D} = \bigvee_{l_r \in K} \neg l_r$$

If the clause $\mathfrak{D}$ is false in an iteration of the DPLL algorithm, this will again lead to the conflict κ. In general, every partition of the node set into a set containing the branching nodes and a set containing the conflict node leads to a conflict clause. State-of-the-art SAT solvers determine the conflict clause based on an *unique implication point* defined as follows (see also [53], [91] and [1]):

Definition 3.2.2. ***unique implication point (UIP)***
*A literal l in the conflict graph G assigned in the decision level d of the search tree is a **UIP**, if every path from the decision nodes at decision level d to the conflict node leads through l or through an UIP at a decision level $d' > d$.*

If every literal which was assigned after the UIP is added to the set N_c and the other literals are added to the set N_r, we call the conflict clause derived from the partition $\{N_r, N_c\}$ a UIP clause. The experiments of Zhang et al. [91] on benchmark problems from microprocessor formal verification and bounded model checking show that the addition of the clause derived from the 1-UIP outperforms approaches using other UIP clauses or other conflict clause generation techniques. In this context, the 1-UIP is the UIP on the highest decision level which is closest to the conflict node κ.

To apply non-chronological backtracking, the CA mechanism also generates 1-UIP *reconvergence* clauses [1]. Therefore, assume that l_d^u is the 1-UIP and d is the

respective decision level. 1-UIP reconvergence clauses are based on a partition of the node set N into N_r which contains all branching nodes and N_c which contains l_d^u but not necessarily κ. Nodes which are on the frontier of N_r, i.e. in the set $R = \{l_r \in N_r \ : \ \exists\, l_c \in N_c \text{ with } (l_r, l_c) \in E\}$ define the respective 1-UIP reconvergence clause $\mathfrak{R}$ as follows:

$$\mathfrak{R} = \bigvee_{l_r \in R} \neg l_r \vee l_d^u$$

Note that the 1-UIP reconvergence clause $\mathfrak{R}$ is also called a clause *not involving a conflict* [91] as here the set N_c may not contain the conflict node κ. In contrast to that, κ must always be an element of the set N_c in a partition $\{N_r, N_c\}$ leading to a conflict clause.

We now consider a special 1-UIP reconvergence clause derived from the partition of N into the sets $\bar{N}_r$ and $\bar{N}_c$ which can be useful for non-chronological backtracking. Assume therefore, that l_d^b is the branching node on the decision level d. Firstly, we add l_d^b to $\bar{N}_r$ and the 1-UIP l_d^u to $\bar{N}_c$. All nodes coming after l_d^b on the paths connecting l_d^b with the 1-UIP l_d^u are inserted into the set $\bar{N}_c$ and the remaining nodes in the set $\bar{N}_r$. The respective 1-UIP reconvergence clause $\mathfrak{R}$ only contains the branching node l_d^b in the form $\neg l_d^b$ from the current decision level d. The other literals of $\mathfrak{R}$ are all of smaller decision levels.

After the deduction of the 1-UIP conflict clause and the special 1-UIP reconvergence clause $\mathfrak{R}$, the CA mechanism is initialized again with the latter clauses. The branching decisions made at smaller decision levels which were involved in the conflict κ together with the two new clauses will lead to the deductions that $l_d^u = 0$ and $l_d^b = 0$. If these new deductions together with the clause database again lead to a conflict, the DPLL algorithm can backtrack to the decision level $d' < d$ where the last branching decision was made which led to the conflict κ. If $d' < d - 1$, the backtracking move is a real backjump, as in contrast to standard backtracking, we resume the algorithm at a decision level which is more than one unit smaller than the current decision level.

With the following example, we aim at illustrating the theoretical concepts from above. It is inspired by the example of Marques-Silva and Sakallah [53].

Example 3.2.1. ***Conflict analysis (clause learning and non-chronological backtracking)***

The following SAT problem is given:

$$\mathfrak{D}_1 = \neg x_1 \vee x_2 \qquad \mathfrak{D}_2 = \neg x_1 \vee x_3 \vee x_7$$
$$\mathfrak{D}_3 = \neg x_2 \vee \neg x_3 \vee x_4 \qquad \mathfrak{D}_4 = \neg x_4 \vee x_5 \vee x_9$$
$$\mathfrak{D}_5 = \neg x_4 \vee x_6 \vee x_{10} \qquad \mathfrak{D}_6 = \neg x_5 \vee \neg x_6$$
$$\mathfrak{D}_7 = \neg x_8 \vee \neg x_{10} \qquad \mathfrak{D}_8 = x_{11} \vee \neg x_7$$
$$\mathfrak{D}_9 = x_{11} \vee x_{12} \qquad \mathfrak{D}_{10} = x_1 \vee x_4 \vee \neg x_{12}$$
$$\mathfrak{D}_{11} = x_{11} \vee \neg x_9 \qquad \mathfrak{D}_{12} = \neg x_{11} \vee \neg x_5 \vee x_6$$
$$\ldots \qquad \ldots$$
$$x_i \in \{0,1\}, \qquad \forall i = 1, \ldots, n$$

As indicated by the dots, the problem also consists of other clauses and variables. We assume, that these are independent of the clauses written above, i.e. they integrate different variables x_i, where $i \neq 1, \ldots, 12$.

The DPLL algorithm makes the following branching decisions:

- $x_8 = 1$ *at decision level 1*
- $x_{11} = 0$ *at decision level 2*
- *At decision level 3 a variable x_i, $i \neq 1, \ldots, 12$ from the remaining problem is chosen*
- $x_1 = 1$ *at decision level 4*

At decision level 4, a conflict κ is detected by UP leading to the conflict graph visualized in Fig. 3.6. Note that the branching decision at level 3 has no effect on the conflict graph, as the remaining problem consists of clauses independent of $\mathfrak{D}_1, \ldots, \mathfrak{D}_{12}$.

x_4 is the 1-UIP, i.e. the UIP on the highest decision level 4 which is nearest to the conflict κ. If we add all literals which were assigned after x_4 to the set N_c and the remaining nodes to the set N_r, we obtain the 1-UIP conflict clause $\mathfrak{C} = x_9 \vee x_{10} \vee \neg x_4$. This clause is added to our problem.

Furthermore, from the partition given by $N_c = \{x_2, x_3, x_4\}$ and the set N_r consisting of the remaining nodes, we obtain the 1-UIP reconvergence clause $\mathfrak{R} = \neg x_1 \vee x_7 \vee x_4$. This clause is also added to our problem.

The re-evaluation of the branching decisions at the levels 1 and 2 again leads to a conflict with the conflict graph given in Fig. 3.7. Thus, both branching decisions $x_1 = 1$ and $x_1 = 0$ at level 4 lead to a conflict and all relevant branching decisions have been made at levels smaller than 3. Hence, instead of continuing with the negated assignment at the decision level 3 as in standard backtracking, the DPLL algorithm can continue with the assignment $x_{11} = 1$ at the decision level 2.

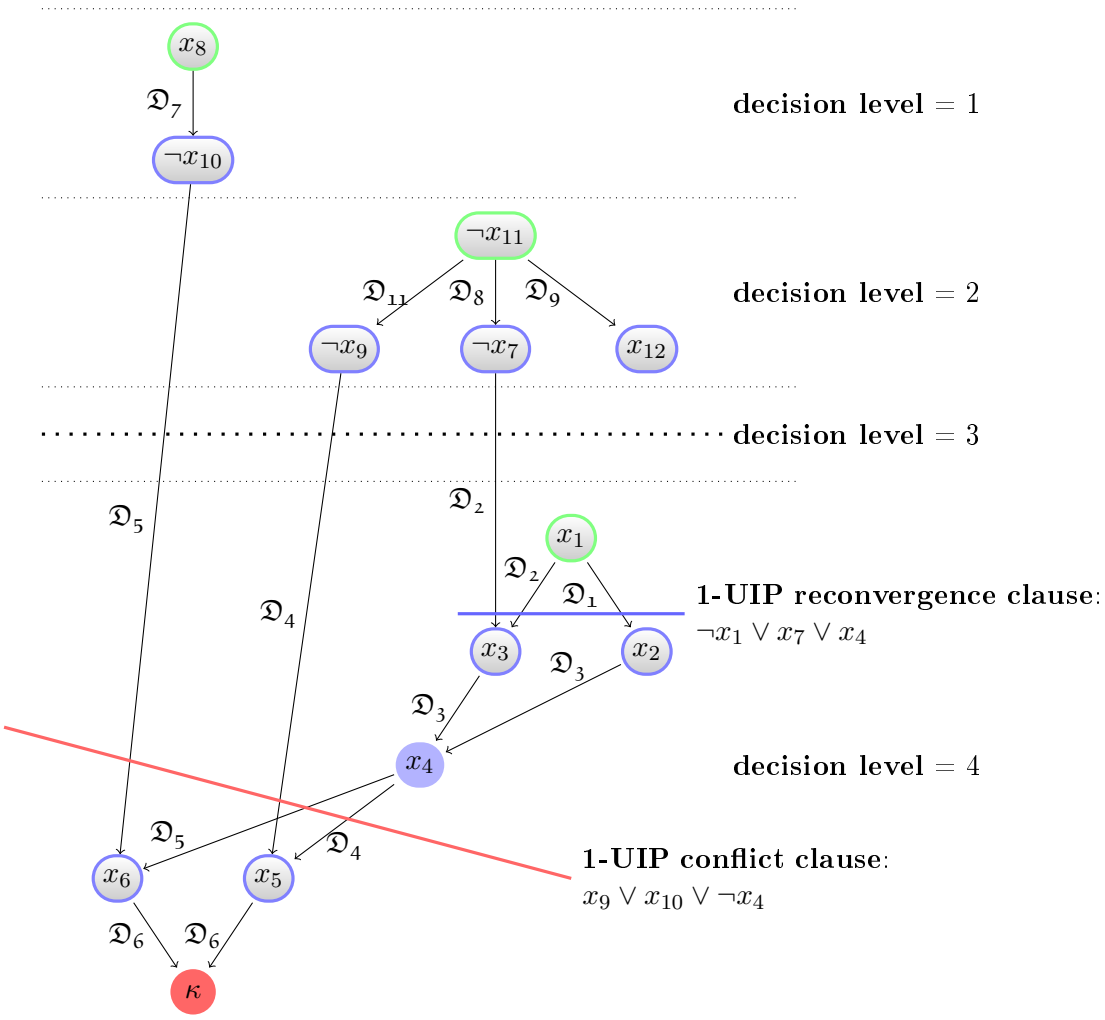

Figure 3.6: Clause learning based on the 1-UIP x_4

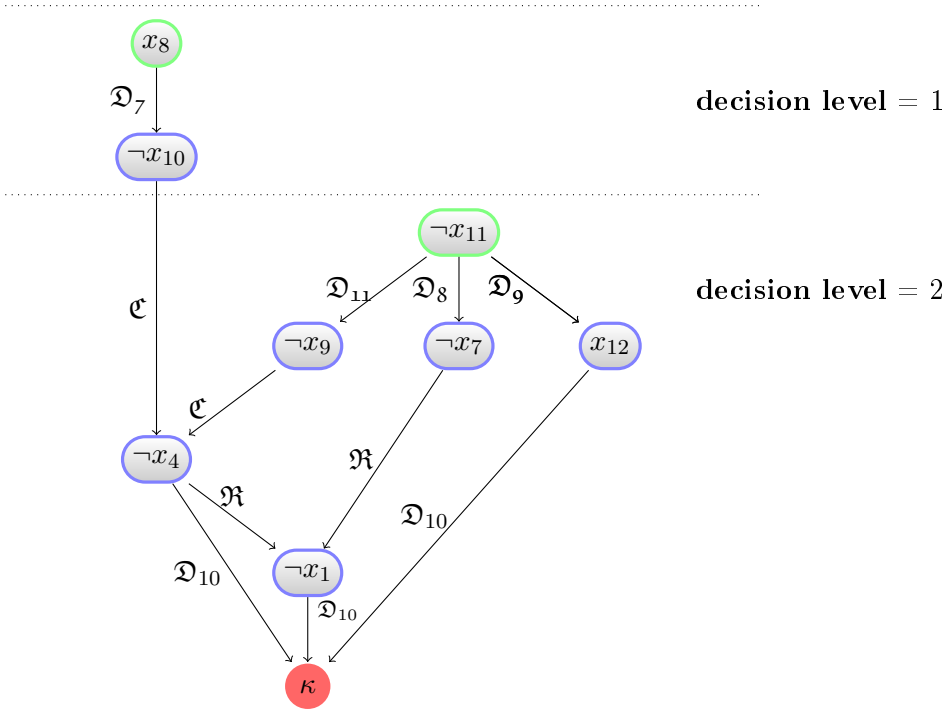

Figure 3.7: Deductions for non-chronological backtracking

3.2.2 Conflict-based variable-value selection strategies and restarts

A well-known variable and value selection strategy for the DPLL algorithm is the variable state independent decaying sum (VSIDS) decision heuristic proposed by Moskewicz et al. [56]. To realize this strategy, the SAT solver maintains conflict counters k_i^0 and k_i^1 for every variable x_i and its respective values 0 and 1 which are initialized with zero.

Whenever a literal appears in a conflict clause, the counter of the respective variable-value combination is increased by one. For example, if the literal $\neg x_i$ is part of a conflict clause, the counter k_i^0 for the assignment $x_i = 0$ becomes $k_i^0 + 1$. The DPLL algorithm selects the unassigned variable and assignment with the highest counter. Ties can be broken by arbitrary strategies. After a predefined period the counters are decreased by a certain factor.

The motivation of this decision heuristic is to fulfill newly generated conflict clauses, i.e. to drive the search away from conflicts. Therefore, the counter of older conflict clauses is also periodically decreased.

Restarting the DPLL algorithm is also a commonly applied technique of SAT solvers. Therefore, the solution process is periodically stopped, i.e. all assignments are undone, and restarted with the conflict clauses and counters from the previous run. The intuition behind this strategy is that in the new run, the SAT solver has potentially more information about literals involved in conflicts. Therefore, the DPLL algorithm can possibly circumvent conflict-prone decisions earlier in the search tree.

The benchmark results of Moskewicz et al. [56] show, that a SAT solver integrating the VSIDS decision heuristic together with restarts and the two watched literal scheme produces on average significantly better results compared to other then state-of-the-art implementations, especially on hard SAT instances.

3.3 Combination of CP and SAT techniques to solve the RCPSP

The BaB algorithm with constraint propagation (see Sect. 3.1) applied in CP solvers can be enhanced by SAT solving techniques, i.e. a CA mechanism, a conflict-based variable-value selection strategies and restarts. The generation of conflict clauses in the context of CP is also known as nogood recording [66].

Two recent optimization frameworks, SCIP [2] and G12 [83], provide a BaB algorithm which combines CP and SAT solving techniques. LCG [62], an exact CP-SAT hybrid integrated into G12, has turned out to be highly efficient for variants of the SRCPSP, i.e. the SRCPSP with SPRs and GPRs and the aim of makespan minimization and the SRCPSP with SPRs and the aim of net present value maximization (see [74], [77] and [75]).

The main approaches used in this dissertation, i.e. LCG and a SCIP-approach are described in Sect. 3.3.1 and 3.3.2, respectively.

3.3.1 Lazy clause generation

LCG was firstly proposed by Ohrimenko et al. [62] and then re-engineered by Feydy and Stuckey [28]. This solution procedure combines a BaB algorithm with constraint propagation and a SAT solver.

Therefore, the domains $\{lb(s_i), \ldots, ub(s_i)\}$ of every variable s_i are encoded by $2 \cdot (ub(s_i) - lb(s_i)) + 1$ boolean variables in the SAT solver:

$$[\![s_i = t]\!] : \; t = lb(s_i), \ldots, ub(s_i)$$
$$[\![s_i \leq t]\!] : \; t = lb(s_i), \ldots, ub(s_i)$$

The variables are true (1) if the respective equations are fulfilled in the recent state of the BaB algorithm and false (0) otherwise. Moreover, clauses have to be defined in the SAT solver about domain based reasoning:

$$\begin{aligned}
[\![s_i = lb(s_i)]\!] &\iff [\![s_i \leq lb(s_i)]\!] \\
[\![s_i = ub(s_i)]\!] &\iff \neg[\![s_i \leq ub(s_i) - 1]\!] \\
[\![s_i = t]\!] &\iff [\![s_i \leq t]\!] \wedge \neg[\![s_i \leq t - 1]\!], && t \in \{lb(s_i) - 1, \ldots, ub(s_i)\} \\
[\![s_i \leq t]\!] &\implies [\![s_i \leq t + 1]\!], && t \in \{lb(s_i), \ldots, ub(s_i) - 1\}
\end{aligned}$$

Furthermore, an assignment of boolean variables in the course of the solution process can be converted back to a domain of a variable s_i, i.e. $domain(s_i)$ consists of the values $t \in \{lb(s_i), \ldots, ub(s_i)\}$ for which

$$\begin{aligned}
&[\![s_i = t]\!] \text{ is true,} \\
\text{or } &[\![s_i \leq t']\!] \text{ with } t' \geq t \text{ is true,} \\
\text{or } &\neg[\![s_i \leq t']\!] \text{ with } t' < t \text{ is true.}
\end{aligned}$$

To integrate clause learning, the constraint propagation algorithms have to provide *explanations* for their processed domain reductions. These are given as clauses to the SAT solvers.

If for example `cumulative` captures TP and EF, it can explain the reasons for inconsistencies or domain reductions in various different ways, for example with a *naive* explanation or a *bound widening* explanation (see also [74] and [72]). An explanation for an inconsistency at the time point t_κ w.r.t. the renewable resource $r \in R$ detected through TP (see (3.9)) is given as follows:

$$\bigwedge_{j : t_\kappa \in cp_j} E_j \implies \texttt{false} \tag{3.16}$$

(5.35) can be divided into the sub-explanations E_j for every job participating in the conflict, i.e. having its compulsory part at time t_κ. In a naive explanation, E_j is given as follows:

$$E_j = \quad [\![lb(s_j) \leq s_j]\!] \wedge [\![s_j \leq ub(s_j)]\!] \tag{3.17}$$

In this context $lb(s_j)$ and $ub(s_j)$ are the lower and upper bounds of the variable s_i at the time the conflict was detected.

In the bound widening explanation E_j is substituted by:

$$E_j = \quad [\![t_\kappa - d_j + 1 \leq s_j]\!] \wedge [\![s_j \leq t_\kappa]\!] \tag{3.18}$$

Note, that the bound widening explanation is potentially stronger than the naive explanation, as $t_\kappa - d_j + 1 \leq lb(s_j)$ and $ub(s_j) \leq t_\kappa$.

It is reasonable to create an explanation for the inconsistency with a minimum number of jobs $j \in J$ with $t_\kappa \in cp_j$ as this leads to a conflict graph with a small number of nodes [72]. Finding the explanation (3.16) with the minimum number of jobs can be done with a time complexity of $(O(|J| \cdot \log(|J|)))$ by sorting the jobs $j \in J$ with $t_\kappa \in cp_j$ in non-increasing order of $c^\rho_{j,r}$ [72]. Jobs $j_1, \ldots, j_k$ are added to the explanation (3.16) based on this sorting until $\sum_{l=1}^{k} c^\rho_{j_l,r} \geq C^\rho_r + 1$ and $\sum_{l=1}^{k-1} c^\rho_{j_l,r} \leq C^\rho_r$.

A lower bound update of job i from $lb(s_i)$ to $t_{\max}$ deduced by TP (see Sect. 3.1.2) w.r.t. resource $r \in R$ is processed and explained in a stepwise manner based on the time points $t_0, \ldots, t_s$ with

$$t_0 = lb(s_i) \tag{3.19}$$

$$t_s = t_{\max} \tag{3.20}$$

$$t_l - t_{l-1} \leq d_i, \ \forall l = 1, \ldots, s \tag{3.21}$$

$$\sum_{j:t_l-1 \in cp_j} c^\rho_{j,r} > C^\rho_r - c^\rho_{i,r}, \ \forall l = 1, \ldots, s \tag{3.22}$$

In general, many different alternatives for the selection of the time points $t_0, \ldots, t_s$ and the jobs j with $t_l - 1 \in cp_j$ exist. The selection of jobs and time points only has to fulfill conditions (3.19) - (3.22). In this context, Schulz [72] shows, that it is strongly $\mathcal{NP}$-hard to find time points $t_0, \ldots, t_s$ fulfilling (3.19) - (3.21) and jobs $j \in J$ fulfilling (3.22) such that the total number of jobs involved in at least one of the s inequalities in (3.22) is minimized.

s explanations E^{lb}_l are generated based on the selected time points. These are of the following form:

$$E^{lb}_l = [\![t_{l-1} \leq s_i]\!] \wedge \bigwedge_{j:t_l-1 \in cp_j} E_j \Longrightarrow [\![t_l \leq s_i]\!], \ \forall l = 1, \ldots, s \tag{3.23}$$

Again, one can distinguish between naive or bound widening explanations as in (3.17), i.e. E_j in (3.23) can be given by:

$$E^{\text{naive}}_j = \quad [\![lb(s_j) \leq s_j]\!] \wedge [\![s_j \leq ub(s_j)]\!]$$

$$\text{or} \quad E^{\text{bw}}_j = \quad [\![t_l - d_j \leq s_j]\!] \wedge [\![s_j \leq t_l - 1]\!]$$

Heinz and Schulz [41] and Schutt et al. [74] provide and evaluate heuristic rules for the choice of the time points $t_0, \ldots, t_s$. Note, that after the choice of the time

points, we can again determine the explanation E_l^{lb} which involves the minimum number of jobs having their compulsory parts in the time interval $[t_l - 1, t_l[$ by a similar principle as in the inconsistency case. Upper bound updates can be explained in a symmetric way.

Example 3.3.1. ***Explanations for TP*** *Assume that we are in the situation of Example 3.1.2. With the time points $t_0 = 3$ and $t_1 = 4$, the naive explanation for the lower bound change of job 1 from 3 to 5 is as follows:*

$$[\![3 \leq s_1]\!] \wedge ([\![2 \leq s_2]\!] \wedge [\![s_2 \leq 4]\!]) \Longrightarrow [\![5 \leq s_1]\!]. \tag{3.24}$$

In this case, the naive explanation is equivalent to the bound widening explanation as $lb(s_2) = 2 = 4 - 3 + 1 = t_1 - d_2 + 1$ and $ub(s_2) = 4 = t_1$.

A lower bound update of the variable s_i deduced by EF can again be explained in a naive way or in a stronger way which applies bound widening [74]. Therefore let $\Psi \subseteq J \backslash \{i\}$ be a set fulfilling (3.13) and $\Psi' \subseteq \Psi$ be a set with $rest(\Psi', i) > 0$. The naive explanation for the update of $lb(s_i)$ to $lb^*(s_i) = lb(\Psi') + \lceil rest(\Psi', i)/c_{i,r}^{\rho} \rceil$ is as follows:

$$[\![lb(s_i) \leq s_i]\!] \wedge \bigwedge_{j \in \Psi} [\![lb(s_j) \leq s_j]\!] \wedge [\![s_j \leq ub(s_j)]\!] \Longrightarrow [\![lb^*(s_i) \leq s_i]\!]$$

Using bound widening the naive explanation from above can be strengthened as follows:

$$[\![lb(s_i) \leq s_i]\!] \wedge \bigwedge_{j \in \Psi \backslash \{\Psi'\}} [\![lb(\Psi) \leq s_j + d_j \leq ub(\Psi)]\!] \wedge \bigwedge_{j \in \Psi'} [\![lb(\Psi') \leq s_j + d_j \leq ub(\Psi')]\!]$$
$$\Longrightarrow [\![lb^*(s_i) \leq s_i]\!]$$

Example 3.3.2. ***Explanations for EF***
In Example 3.1.3 the lower bound of job 1 is updated to $lb^{\max}(s_1) = 3$. The update is based on the set of jobs $\Psi(\Psi') = \{2, 3\}$. Therefore the naive explanation is as follows:

$$[\![1 \leq s_1]\!] \wedge ([\![1 \leq s_2]\!] \wedge [\![s_2 \leq 2]\!]) \wedge ([\![1 \leq s_3]\!] \wedge [\![s_3 \leq 4]\!]) \Longrightarrow [\![3 \leq s_1]\!]$$

Again, the bound widening explanation is equal to the naive explanation from above. If e.g. the upper bound of job 2 $ub(s_2)$ is equal to one, this would lead to the same lower bound update of job 1, but the bound widening explanation would be stronger than the naive explanation, because then $ub(s_2) = 1 < 2 = ub(\{2, 3\}) - d_2 = 2$.

State-of-the-art EF algorithms as proposed in [88] and [55] do not compute the sets Ψ and Ψ', for which $lb^{\max}(s_i)$ in (3.14) is attained, explicitly. To store information about the sets Ψ and Ψ' during the EF algorithms, Schutt et al. [74] extend the state-of-the-art EF algorithms which leads to an increase in complexity by a factor of $|J|$.

In the SCIP-approach of [72] the values $lb(\Psi')$, $ub(\Psi')$, $lb(\Psi)$ and $ub(\Psi)$ are stored in the course of the EF algorithm. Hence, in case the lower bound update of s_i to $lb^{\max}(s_i)$ is part of explanations leading to a conflict, the latter information can be accessed to compute the sets Ψ and Ψ'. More precisely, Ψ and Ψ' consist of jobs j with $lb(\Psi) \leq s_j + d_j \leq ub(\Psi)$ and $lb(\Psi') \leq s_j + d_j \leq ub(\Psi')$ respectively at the time the propagation took place.

The explanations generated by the global constraints are given as clauses to the SAT solver. The boolean literals on the left hand side of the explanation, i.e. before $\Longrightarrow$, are assigned to 1 (*true*). Thus, the SAT solver deduces by UP, that also the literal on the right hand side of the explanation, i.e. after $\Longrightarrow$, must be true. In case of a conflict, the CA techniques of the SAT solver computes nogoods based on the resulting conflict graph[2]. These nogoods are added to the clause database of the SAT solver to possibly deduce new domain reductions or cutoffs in the BaB tree. Note, that in the current implementation of LCG described in [28], non-chronological backtracking is not supported.

Furthermore, the conflict statistics of the SAT solver guide the branching decisions in the BaB algorithm. More precisely, the variable-value combination with the highest conflict counter is chosen, i.e. if the literal $[\![y \leq t^*]\!]$ has the highest conflict counter, two new nodes are created, where $y \leq t^* \quad$ or $\quad y \geq t^* + 1$. In other words, LCG integrates the VSIDS branching heuristic. The computational experiments of Schutt et al. [74] and Schutt et al. [77] on SRCPSP instances with SPRs and GPRs respectively show the benefit of a conflict-driven search strategy with periodical restarts over a scheduling-specific search strategy.

3.3.2 The optimization framework SCIP

The optimization framework SCIP integrates solution techniques from CP, MIP and SAT solving and combines them in a Branch and Bound (BaB)-algorithm with preprocessing [1]. In addition, it can be used as a standalone CP, MIP and SAT solver. Moreover, the user can influence the SCIP-intern solution procedure by adding new constraint handlers (i.e. global constraints), primal heuristics, cutting plane techniques, branching rules etc. [2].

The constraint handler `cumulative` was introduced by Berthold et al. [11] for SCIP. This constraint handler captures TP and EF algorithms with a connection to the SCIP-intern CA mechanism through explanation generation. The explanation generation principles are described in [72]. These are similar to the ones introduced in Sect. 3.3.1.

SCIP also integrates a conflict-driven branching rule called *inference branching* [1]. The latter integrates ideas from the VSIDS decision heuristic. Because of the functionality provided by SCIP, one can apply this tool to solve the SRCPSP with

[2] An good example for the CA process in the context of LCG for the SRCPSP can be found in [77].

an approach similar to LCG. Computational experiments testing the solution of the SRCPSP with SPRs with SCIP are summarized in Sect. 4.3.

An important difference between the solution approach of SCIP and LCG, which is usable via G12, is the principle of the solver-intern CA mechanism. LCG generates explanations and adds them to the SAT solver every time a global constraint deduces a bound update or a conflict. In SCIP, the CA mechanism is initialized only when a conflict is detected by a constraint handler. After the initialization, the CA mechanism searches for variables and respective bounds which are involved in the conflict. If a new bound of a variable derived by a certain constraint handler is involved in a conflict and the respective constraint handler integrates explanation generation, the CA mechanism uses this explanation of the bound update to generate a conflict graph. If the conflict graph is fully initialized, the CA mechanism searches for no goods and backjumps. Informally speaking, in LCG the conflict graph is constructed in a forward manner, whereas the conflict graph in SCIP is generated in a backward manner. Advantages and disadvantages of the conflict graph construction strategies are discussed in [41].

In this context, it is also important to note, that SCIP is the first solver to integrate CA in the context of MIP, i.e. it incorporates conflict learning based on infeasible linear programming relaxations [1].

3.3.3 Conclusion on optimization frameworks

To conclude, there are two important optimization frameworks relevant in this dissertation, which incorporate solution principles from CP and SAT solving, namely SCIP and G12. SCIP offers more flexibility to the user, as there is e.g. the possibility to implement new constraint handlers for specific problem characteristics and to directly influence the solution algorithm. In G12, the user can only use the existing global constraints to formulate a problem at hand in the modeling language Zinc (see [54]). However, it is possible to choose between different exact solution algorithms including LCG, the state-of-the-art exact approach for variants of the SRCPSP.

Chapter 4

Performance variability of CP-SAT solvers for the RCPSP

In this chapter we analyze the effect of the predefined search space on recent exact CP-SAT algorithms for the SRCPSP with SPRs. Firstly, Sect. 4.1 provides a well-known CP model for the SRCPSP with SPRs on which our evaluation is based. For our computational experiments, we use the optimization frameworks SCIP [2] and G12 [83], i.e. solvers which provide a solution procedure combining CP and SAT Solving techniques (see Sect. 3.3). Secondly, Sect. 4.2 presents the procedure to construct the predefined search space and describes our parallel algorithm to benefit from the performance variability which results from varying the initial search space. Finally, Sect. 4.3 shows the architecture and results of our computational experiments on instances with 60 jobs. Thereby, Sect. 4.3.1 emphasizes the effect of the predefined search space on the analyzed exact algorithms and Sect. 4.3.2 shows a comparison of our parallel procedure to recent exact algorithms. Finally, in Sect. 4.3.3 we explain the observed effect on the basis of further computational experiments. The chapter ends with a conclusion derived from the obtained results.

4.1 A CP model for the SRCPSP with SPRs

Using the integer variables s_j for the modeling of the starting time of job j, the SRCPSP with SPRs can be modeled as follows in chosen CP modeling languages:

$$\begin{aligned}
&\min && s_{n+1} && && (4.1)\\
&s.t. && s_i + d_i \leq s_j && \forall\, j \in S_i, \forall\, i \in J && (4.2)\\
&&& \texttt{cumulative}(\mathbf{s}, \mathbf{d}, \mathbf{c}_r^\rho, C_r^\rho) && \forall r = 1, \ldots, m && (4.3)\\
&&& s_j \in \{0, \ldots, T\}, && \forall j \in J && (4.4)
\end{aligned}$$

Our aim is the minimization of the makespan which equals the starting time s_{n+1} of the dummy job $n+1$ (see Sect. 2.1.3). Moreover, T is a known upper bound on the project makespan s_{n+1} and can e.g. be set to $\sum_{i \in J} d_j$.

For the formulation of the resource constraints, we use the CP scheduling constraint `cumulative` (see also Sect. 3.1.2). This constraint guarantees resource feasibility for a certain resource $r = 1, \ldots, m$:

$$\sum_{j \in J, s_j \leq t < s_j + d_j} c_{j,r}^{\rho} \leq C_r^{\rho}, \quad \forall t \in 0, \ldots, T-1 \tag{4.5}$$

Thus, the complete usage of resource r in time period $[t, t+1]$, $0 \leq t \leq T-1$ must not exceed the maximal capacity C_r^{ρ}.
If we implement the model within SCIP or G12, the `cumulative`-constraint captures scheduling-specific propagation and explanation generation algorithms (see Sect. 3.3).

4.2 Activity List Preprocessing and Independent Parallel Solution

In the computational experiments of Sect. 4.3, we analyze the effect of the predefined variable domains on the behavior of recent exact algorithms. Before that, this section outlines our parallel procedure to exploit the positive effect of varying initial domains on the outcome of the exact algorithms. As the preprocessing step of our procedure mainly consists of the random generation of a number of precedence feasible permutations of jobs, i.e. activity lists (see Sect. 2.5), we call it AL preprocessing.

To reduce the search space, one can define feasible lower and upper bounds $lb(s_j)$ and $ub(s_j)$ for the starting times (see also Sect. Sect. 2.4). Given an upper bound (UB) on the makespan s_{n+1}, which is the starting time of the dummy job $n+1$, one can determine $lb(s_j)(ub(s_j))$ by forward (backward) recursion (FBR) [16]:

$$lb(s_0) = 0; \qquad lb(s_j) = \max_{i \in P_j} \{lb(s_i) + d_i\}, \; j \in J - \{0\} \tag{4.6}$$

$$ub(s_{n+1}) = \mathrm{UB}; \quad ub(s_j) = \min_{i \in S_j} \{ub(s_i) - d_j\}, \; j \in J - \{n+1\} \tag{4.7}$$

The above procedure for determining feasible domains for the variables s_j is only based on the precedence constraints (4.2). Thus, $lb(s_j)$ is the maximum of the earliest finishing times of the direct predecessors P_j of job j and the latest finishing time is the minimum of the latest starting times of the direct successors S_j of j.

Once having determined $lb(s_j)$, $ub(s_j)$ for an upper bound UB by (4.6) and (4.7), $lb'(s_j)$, $ub'(s_j)$ can be easily evaluated for another feasible upper bound UB':

$$lb'(s_j) = lb(s_j) \tag{4.8}$$

$$ub'(s_j) = ub(s_j) - (\mathrm{UB} - \mathrm{UB}') \tag{4.9}$$

Firstly, via the AL preprocessing step, we generate a number of feasible upper bounds $\mathrm{UB}_1, \ldots, \mathrm{UB}_l$. Afterwards, the upper bounds UB_k, $k \in 1, \ldots, l$ are used for

the determination of $lb^k(s_j)$ ($ub^k(s_j)$) by (4.6)-(4.9). Finally, for every $k \in 1,\ldots,l$, we define the CP-model RCPSP^k of Sect. 4.1 with the variable domains $D_j^k = \{lb^k(s_j),\ldots,ub^k(s_j)\}$ and solve the models RCPSP^k, $k \in 1,\ldots,l$ independently on l different processors by the solver tools implementing the general exact solution procedures described in Sect. 4.1.
For the generation of feasible upper bounds $\mathrm{UB}_1,\ldots,\mathrm{UB}_l$, we first determine $lb(s_0)$ and $ub(s_0)$ based on the trivial upper bound $\mathrm{UB}_0 = \sum_{j\in J} d_j$. Then we generate a number of ALs by a randomized version of the minimum latest finishing time priority rule (MINLFT).

Given a set of eligible activities E_u (i.e. activities for which all predecessors have already been inserted in the current AL), we randomly choose the next activity $j_u \in E_u$ for the insertion with the following probability (see also [80]):

$$pr_{j_u} = \frac{\max_{k\in E_u}\{ub(s_k)+d_k\} - (ub(s_{j_u})+d_{j_u})}{\sum_{e\in E_u}(\max_{k\in E_u}\{ub(s_k)+d_k\} - (ub(s_e)+d_e))} \tag{4.10}$$

After having generated an AL $A_k = (j_0,\ldots,j_{n+1})$, we can generate a feasible upper bound UB_k with the SSGS (see [47] and Sect. 2.5). Thereby, SSGS takes the activities j_u, $u = 0,\ldots,n+1$ from A_k in the given order and schedules them at the earliest point in time such that resource and precedence feasibility is guaranteed. This procedure can be implemented with a running time of $O(n^2\,m)$ (see [6] or [16]), where m is the number of resources. Finally, we obtain $\mathrm{UB}_k = s_{n+1}^k$.

It has to be mentioned that two different AL can lead to the same upper bound UB. Clearly, for the generation of the domains $D_j^k = \{lb^k(s_j),\ldots,ub^k(s_j)\}$ we only consider UBs with $\mathrm{UB}_k \neq \mathrm{UB}_s$, $\forall k,s$ with $k \neq s$.

Fig. 4.1 again summarizes our AL preprocessing and parallel exact solution procedure (ALPaPES). For the results presented in Sect. 4.3.1 we omit Step 2b), i.e. we wait for the results of every processor such that the advantage of the parallel solving runs with different domains D_j^k can be shown.

4.3 Computational Experiments

The following section summarizes the computational results of the ALPaPES procedure of Sect. 4.2. Thereby, the results are based on evaluations with the 480 60-job instances of the PSPLIB introduced by Kolisch and Sprecher [50][1].

On the one hand, we want to show that our procedure can improve recent general exact algorithms. Therefore, we compare our results to the LCG and the SCIP approach (see Sect. 4.3.2). On the other hand, we want to show the positive effect of generating more AL and using more processors (see Sect. 4.3.1).

[1] We also carried out evaluations on the 90- and 120-job instances from the PSPLIB which are beyond the scope of this dissertation. These experiments reinforce the results of the following sections.

Input Parameters: Number p of ALs to generate and time limit L

Step 1: AL Preprocessing

a Randomly generate p feasible ALs based on a fixed priority rule (see e.g. (4.10)).

b Apply the SSGS to the ALs to obtain l different upper bounds $\mathrm{UB}_1, \ldots, \mathrm{UB}_l$ on the makespan.

c Determine the domains D_j^k for all $j \in J$ and $k = 1, \ldots, l$ based on (4.8) and (4.9) with $lb^0(s_j)$ and $ub^0(s_j)$ calculated from $\mathrm{UB}_0 = \sum_{j \in J} d_j$ in Step 1a.

Step 2: Parallel exact solution

a Solve the models RCPSP^k defined by the domains $D_j^k, j \in J$ by a general exact solution procedure on processor k with the time limit L.

b Stop the solution process on all processors with $k \neq u$ as soon as the first model RCPSP^u is solved optimally or stop after the time limit L has elapsed.

Output: Optimal makespan s_{n+1}^* or upper bound $s_{n+1}^{'} = \min_{k=1,\ldots,l} \left\{ s_{n+1}^k \right\}$.

Figure 4.1: AL preprocessing and parallel exact solution (ALPaPES)

Also in Sect. 4.3.1, we computationally verify that the best results of our approach are mostly not achieved through the smallest search space determined through $UB^* = \min_{k\in\{1,\dots,l\}}\{UB_k\}$. The experiments of Sect. 4.3.3 give an explanation for the observed effect.

The experiments were carried out on the Vienna Scientific Cluster. Thereby, the cluster nodes integrate a X86-64 architecture running under GNU/Linux with Intel(R) Xeon(R) X5650 processors of 2,66GHz and with 24GB RAM. The experiments with SCIP were carried out with version 3.0.0 and the additional branching rule presented in Sect. 4.3.2 which was implemented by the SCIP team.

For the experiments with LCG we used the G12 Constraint Programming Platform (see [83]) (version 2.0.0) provided by the NICTA research group. Thereby, we used the Zinc-modeling language with a mapping to the g12_fdx LCG solver which uses MiniSat as the underlying SAT-Solver. A time limit $L = 600\,\text{s}$ was used for the solver runs as in the LCG-approach of [74]. For both, the SCIP and the G12 runs, we implemented the CP-model presented in Sect. 4.1 with different variable domains obtained through the UBs.

For the parallel SCIP runs, we used Open MPI 1.4.3 (see [32]) in combination with the C++-programming language. The parallel G12 runs were carried out with the Python-module mpi4py (see [57]) in combination with the Python programming language.

4.3.1 Positive Effects through the Generation of more Activity Lists

For the following experiments, we generated a maximum of 40 AL for each of the 480 60-job instances which lead to an average and maximum number of 9.16 and 24 different UBs, i.e. processors used for the parallel solving runs.

The results of our ALPaPES procedure on all 480 60-job instances from the PSPLIB can be found in the Tables 4.1 and 4.2. There, we can see

1. the number of instances solved to optimality (# opt)
2. the number of times we reached the best known solution from the PSPLIB (# best)
3. the average solution time (in seconds)
4. the average solution time on the instances solved to optimality (and solved to optimality in the 40 AL setting)
5. the average relative gap w.r.t. the best known solutions (PSPLIB) (%)
6. the average gap w.r.t. the critical path lower bound (%)[2]

[2]Given through the length of the longest path in the successor-predecessor network.

when generating 1, 4 and 40 AL and when we use the lowest UB given through the 40 generated AL (Lowest UBM) for the solver runs. Thereby, the solution time is the CPU-time measured by SCIP and G12.

The measurements regarding 1, 4 and 40 AL correspond to the best results obtained by SCIP or G12 on the respective number of processors.

Table 4.1: Comparison on all 480 instances (SCIP)

Measurements SCIP	**# AL**			**Lowest UBM**
	1	**4**	**40**	
# opt	420	421	425	423
# best (PSPLIB)	425	430	434	425
avg.sol.time (all Inst.)	80.82	79.11	74.42	79.79
avg.sol.time opt.(opt.comp.)	6.65 (13.63)	6.12 (11.70)	6.40	9.67 (12.47)
avg.gap (best)	0.61	0.51	0.37	0.59
avg.gap (crit. path LB)	11.34	11.20	11.00	11.32

Table 4.2: Comparison on all 480 instances (G12)

Measurements G12	**# AL**			**Lowest UBM**
	1	**4**	**40**	
# opt	430	430	430	429
# best (PSPLIB)	433	433	442	435
avg.sol.time (all Inst.)	68.97	67.70	66.79	69.94
avg.sol.time opt.(opt.comp.)	7.22 (7.22)	5.81 (5.81)	4.79	6.92 (8.30)
avg.gap (best)	0.42	0.30	0.21	0.46
avg.gap (crit. path LB)	11.06	10.87	10.71	11.12

We can observe the following from the Tables 4.1 and 4.2:

 # opt (best) The results for the SCIP runs show that it is better to generate more AL and use more processors. We can solve 5 more instances to optimality and we attain 9 more best known solutions compared to using only 1 AL. Moreover, we can see that the results with 40 AL outperform the results when using the lowest UB. Nevertheless, regarding # opt, performing the SCIP runs with the lowest upper bounds should be preferred to running our ALPaPES procedure with 1 or 4 AL.
When we regard the results of the G12 runs, we can observe nearly the same regarding # best. The best known results are attained 9 times more often than when we use only 1 AL.
Regarding # opt it does not make any difference to use 1,4 or 40 processors. We can solve 5 more instances to optimality than with SCIP. Surprisingly,

Table 4.3: Sol.time comparison on the non-easy instances

Measurements	**# AL**			**Lowest**
SCIP	**1**	**4**	**40**	**UBM**
avg.sol.time	125.63	107.92	58.97	114.96
max.diff sol.time (40 AL)	596.37	446.01	-	465.68

Measurements	**# AL**			**Lowest**
G12	**1**	**4**	**40**	**UBM**
avg.sol.time	46.29	37.03	30.28	53.55
max.diff sol.time (40 AL)	197.65	79.92	-	492.85

when using only 1 randomly generated AL, we can solve 1 more instance to optimality than with the lowest of the 40 UB.

Solution Time Regarding the average solution time for all instances and the instances solved to optimality, we can see that the setting with 40 AL clearly outperforms all other settings both in the SCIP and G12 runs. Moreover, in the setting with the lowest UB, the average solution time is ≈ 2 times higher than in the SCIP runs with 40 AL on the 425 instances solved to optimality. In the G12 runs, this factor is ≈ 1.7. Moreover, the second worst and the worst average results are obtained in the setting with the lowest UB for SCIP and G12, respectively.

Gap Again, for the SCIP and the G12 runs, the best results can be detected in the 40 AL setting. In the G12 runs, the worst and in the SCIP runs the second worst results are obtained with the lowest UB. Moreover, when generating 40 AL and applying our ALPaPES procedure with G12 (SCIP), we obtain a $\approx 50\%$ (40%) better relative gap w.r.t. the best known solution from the PSPLIB.

In the above evaluations the use of several processors seems to have only a little effect on the average solution time w.r.t. all 480 instances. This is due to the fact, that a great percentage of the 480 60-job instances can be considered as easy, i.e. they can be solved by SCIP or G12 in less than 1 second. To emphasize the gain of using more processors regarding solution time, we excluded easy instances from our evaluations, i.e. instances where the maximal solution time of all processors in the 40 AL setting is smaller than 1 second for the used solver.

With SCIP, we could eliminate 379 and with G12, we could eliminate 365 instances. The following tables and figures are based on the remaining 101 and 115 instances, respectively.

Table 4.3 shows the results regarding solution time on the 46 and 65 non-easy instances which can be solved to optimality by SCIP and G12, respectively. There, the gain of generating more AL and the parallel solution becomes even clearer. We obtain an average improvement of 66.7s and 16.0s in the 40 AL setting compared to

Table 4.4: Gap comparison on the open instances

Measurements SCIP (best PSPLIB)	**# AL**			**Lowest UBM**
	1	**4**	**40**	
avg.gap	5.23	4.46	3.38	5.13
max.dist gap	17.9	17.9	14.3	17.4

Measurements G12 (best PSPLIB)	**# AL**			**Lowest UBM**
	1	**4**	**40**	
avg.gap	4.0	2.8	1.9	4.3
max.dist gap	14.9	7.4	5.2	9.8

the setting with 1 AL with SCIP and G12, respectively. The lowest UBM setting leads to the worst results with G12. In this setting, the average solution time is by 56.0s and 23.3s higher than in the best setting (40 AL) with SCIP and G12, respectively. Additionally, we can achieve a maximal improvement in solution time of 596.37s and 492.85s with SCIP and G12, respectively when using the best found search space[3] for the solver runs.

Table 4.4 shows the results regarding the average gap to the best known solutions for the 55 and 50 instances which could not be solved to optimality by SCIP and G12, respectively. We can see that in the 40 AL setting, we can reduce the maximum relative gap by 3.6% and 9.7% with SCIP and G12, respectively.
Finally, we want to show the versatile distribution of the relative difference of the UBs leading to the best results (UB_k^{Best}, $k \in K$) w.r.t. the lowest of the 40 UBs (UB_k^*, $k \in K$) for the set of non-easy instances w.r.t. SCIP ($K = \{1, \ldots, 101\}$) and G12 ($K = \{1, \ldots, 115\}$):

$$\Delta_k^{Best} = \frac{\mathrm{UB}_k^{Best} - \mathrm{UB}_k^*}{\mathrm{UB}_k^*} \tag{4.11}$$

The histograms in Fig. 4.2 and 4.3 plot $\left|\left\{k \in K : a \leq \Delta_k^{Best} < b\right\}\right|$ for different values $a \geq 0$ and $b \leq 0.30$ for SCIP and G12.

The best found search space comes from Δ_k^{Best}-values in a range of $0-26.0\%$ and $0-20.1\%$ w.r.t. the lowest UB of the considered instance for the SCIP and G12 runs. Thus, it is not worthwhile to consider UBs with $\mathrm{UB}_l > 1.26 \cdot \mathrm{UB}_l^*$ for an instance l for the solver runs with the 60 jobs instances. Moreover, for a great portion (76% and 82%) of the instances, Δ_k^{Best} lies within the range of $0-10\%$. However, in most cases (70% and 78%), the smallest Δ_k^{Best} does not lead to the best solver runs.

The above evaluations clearly show the gain of generating more AL and therefore using more processors. In the 40 AL settings, we achieved the best results. Moreover, the efficiency of the applied exact solving algorithm strongly depends on the predefined search space given through the UBs. By taking the best found search

[3]This search space is given through the variable domains leading to the best results on a processor.

space (40 AL setting) which is in most cases not given through the lowest UB, we achieve a great improvement regarding the average solution time and the relative gap to the best known solutions. This also emphasizes the increase in efficiency of our ALPaPES procedure through the use of more processors.

4.3.2 Comparisons to Recent General Exact Approaches

In this section, we compare the results of our ALPaPES procedure with recent exact algorithms whose principles are described in Sect. 4.1. At first, we compare our ALPaPES procedure with 40 AL using SCIP (ALPaPES-SCIP) with the state-of-the-art SCIP approach of [11] (SCIPN) in Table 4.5.

The enormous improvement of ALPaPES-SCIP compared to SCIPN concerning solution time and quality is due to the following aspects:

1. The MIP techniques applied in the `cumulative`-constraint have a rather high computational overhead compared to their impact on the dual bound. When switching off all MIP-Plugins in SCIP and using SCIP as a standalone CP+SAT Solver, additional evaluations show that # opt can be increased by 12 compared to SCIPN.

2. By a new conflict-driven branching rule in combination with SCIP 3.0.0 # opt can be increased by 17. In all existing SCIP versions, the conflict statistics used by the inference branching rule (see [1]) are only connected to the variables s_i but not to their domains. The variable s_i with the highest conflict-based score is chosen and the following two branches are created (if the domain of s_i has more than one value):

$$s_i \leq \left\lfloor \frac{lb(s_i) + ub(s_i)}{2} \right\rfloor \quad s_i \geq \left\lfloor \frac{lb(s_i) + ub(s_i)}{2} \right\rfloor + 1 \tag{4.12}$$

   The new conflict-driven branching rule which we applied for our experiments integrates conflict statistics based on variables and domains. Thereby, the conflict-based score is measured and updated for every boolean variable $[\![s_i \leq t]\!]$, $[\![s_i \geq t]\!]$ with $lb(s_i) \leq t \leq ub(s_i)$. Branching is applied based on the boolean variable with the highest score.

3. Finally, our ALPaPES procedure leads to the improvement which is described in Sect. 4.3.1. This improvement can be achieved through generating a number of different UBs as input for parallel SCIP runs instead of using only 1 run with a random UB.

Finally, in Table 4.6 our ALPaPES procedure with 40 AL using G12 (ALPaPES-G12) is compared to the best G12 approach (G12NRg) given by [74][4].

[4]Note that [74] used LCG via the Mercury programming language, not by a call of the g12_fdx LCG solver via the Zinc modeling language like in our case.

Thereby, G12NRg is a LCG-approach with restarts of the VSIDS search of the underlying SAT solver and with explanations obtained through the timetable propagation algorithm of the `cumulative`-constraint.

The average solution time of ALPaPES-G12 is approximately the same compared to G12NRg. The gain of ALPaPES-G12 can be detected when we analyze the obtained solution quality. In nearly the same time we reach 442 best known solutions compared to the 436 best known solutions of G12NRg. Moreover, we can decrease the average distance to the best known solutions w.r.t. the open instances by 1.46 units.

All in all, with our ALPaPES procedure we clearly outperform the state-of-the-art SCIP approach of Berthold et al. [11] and we are competitive with the state-of-the-art exact approach of [74] and even improve it w.r.t. the obtained solution quality.

4.3.3 Effect of a different branching rule

In the following, we present the results of a more simple branching rule in combination with the solver G12 to computationally explain the above effect.

This rule chooses an (unassigned) variable based on a fixed order and creates a new branch by setting the variable to its actual lower bound. The results are outlined in Table 4.7.

We can see that the use of this branching rule leads to clearly inferior results regarding solution time and quality. Only 397 instances can be solved to optimality compared to the 430 instances when using the VSIDS decision heuristic. Moreover, in most cases ($\approx 80\%$) the lowest UB setting leads to the best results (see Fig. 4.4). Firstly, we can conclude that the VSIDS decision heuristic is a key factor leading to the efficiency of the LCG-approach. Moreover, we can not observe a similar variation in the outcome of the LCG approach as with the VSIDS decision heuristic. Thus, the observed variation in the exact algorithms can be explained by the effect of the predefined variable domains on the branching decisions obtained through VSIDS. Variations in the UBs of the variable domains can lead to different values in the conflict statistics. At a point in the BaB tree this can lead to completely different decisions for different initial domains. Varying decisions can infer a more successful exploration of the search space, i.e. an optimal solution can be found earlier or stronger nogoods or greater backjumps can be deduced. Both cases can lead to an improvement in solution time and quality and thus to a better outcome with a higher UB as input.

4.4 Conclusion

We have analyzed the effect of the predefined search space on the behavior of recent exact algorithms for the SRCPSP with SPRs (see [11] and [74]). Moreover, we have

developed a simple parallel solving procedure to take advantage of the positive effects found in our investigation.

Our computational results show, that the efficiency of the applied solvers (SCIP and G12) strongly depends on the predefined variable domains which define the search space. We observed variations in solution times of 582s and an increase of the maximal gap to the best known solution by 9.7% when a "bad" predefined search space is used for the solver runs. Moreover, the predefined search space leading to the best performance of the underlying solver was rarely given by the variable domains determined through the lowest UB.

Furthermore, we computationally verified that the above effect is mostly due to the influence of the initial variable domains on the branching decisions based on VSIDS. Varying initial domains can lead to completely different conflict statistics which guide the path in the BaB search tree. Thus, larger initial variable domains can lead to a more successful exploration of the search space, i.e. the exact algorithm is able to find the optimal solution earlier or to derive stronger nogoods or greater backjumps. As it is hard to predict the effect of the predefined search space on the branching decisions in advance, parallel computing was our method of choice. We developed a simple parallel procedure (ALPaPES) to take advantage of the positive dimensions of the measured effect. This procedure consists of running an exact algorithm on the same instance with different predefined upper bounds on different processors. Thereby, the best result of all used processors determines the outcome of our procedure. The gain of our procedure is clearly due to the use of parallel computing. Sequential testing of the generated UBs would not be efficient.

By our ALPaPES procedure with SCIP as the underlying solver, we can significantly outperform the state-of-the-art SCIP approach of [11] on the 60-job instances of the PSPLIB. We decrease the average solution time by nearly 92% and solve 34 more instances to optimality.

Furthermore, when using G12 as the underlying solver, we are competitive with the state-of-the-art exact approach of [74] considering solution time and even outperform this approach w.r.t. solution quality. With our approach we reach the best known solutions on the 60-job instances from the PSPLIB in 6 more cases and we decrease the average distance to the UBs of the PSPLIB by 1.46 units.

An interesting future direction is a theoretical analysis of the observed effect. The main question to pose is if there is a possibility for the exploitation of the effect without the use of parallel computing. Another point is the analysis of other initial factors influencing the exact algorithms and to exploit them in a parallel environment.

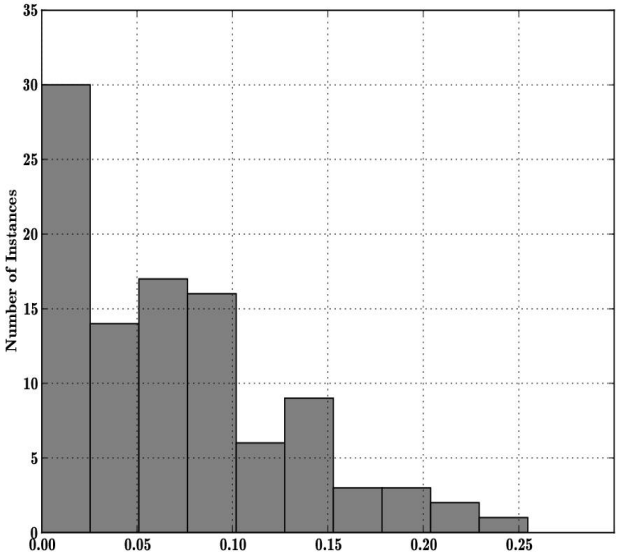

Figure 4.2: Distribution of the Δ_k^{Best}-values (SCIP)

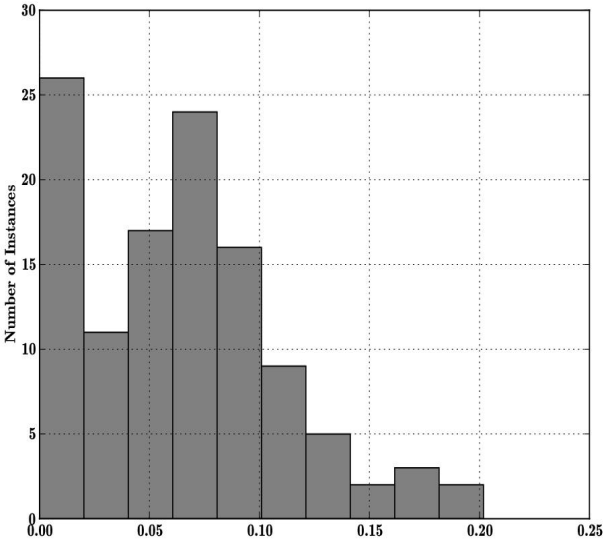

Figure 4.3: Distribution of Δ_k^{Best}-values (G12)

Table 4.5: Comparison with recent exact approaches

	avg.sol.time (480)	**avg.s.t. (opt.)**	**#opt**	**#best**
SCIPN	892.10	358.60	391	**401**
ALPaPES-SCIP	74.42	6.40	425	**434**

Table 4.6: Comparison with recent exact approaches

	avg.s.t.(480)	**avg.s.t. (opt.)**	**#opt**	**#best**	**avg.dist.b.**
G12NRg	66.01	4.01	430	436	3.7
ALPaPES-G12	66.79	4.79	430	442	2.24

Table 4.7: Results with the branching rule: Smallest variable index, smallest domain value

G12	**# AL**			**Lowest UBM**
	1	**4**	**40**	
# opt	393	394	397	395
# best (PSPLIB)	393	394	397	395
avg.sol.time (all Inst.)	112.0	111.21	110.24	111.46
avg.sol.time opt.(opt.comp.)	3.97 (9.97)	4.52 (9.03)	7.84	6.34 (9.33)
avg.gap (best)	3.94	4.04	3.11	3.18
avg.gap (crit. path LB)	17.5	16.3	15.0	15.1

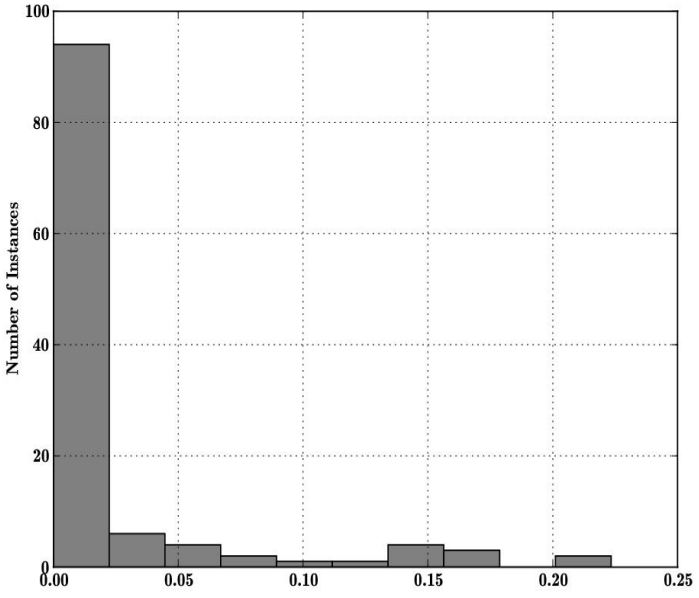

Figure 4.4: Distribution of Δ_k^{Best} for the new branching rule

Chapter 5

CP-SAT approaches for the MRCPSP with SPRs

In this chapter, extensions of the state-of-the-art CP-SAT approaches for the SR-CPSP with SPRs are proposed with which multi-mode instances of the latter problem can be solved in an efficient way. Sect. 5.1 introduces three problem formulations in optimization frameworks which support the solution by a BaB algorithm integrating CP and SAT techniques. One of these models integrates `cumulativemm`, a new constraint handler which we implemented within SCIP. In Sect. 5.2, we describe the principles of the latter. Sect. 5.3 discusses the results of our computational experiments and draws a comparison to the state-of-the-art exact approach for the MRCPSP with SPRs of Zhu et al. [92]. The chapter ends with a conclusion derived from the obtained results.

5.1 G12- and SCIP-models for the MRCPSP with SPRs

In the following, we present three CP-models for the MRCPSP with SPRs in the optimization frameworks G12 and SCIP (see Sect. 3.3). Sect. 5.1.1 contains one model which can be used within the Zinc-modeling language [54] which is supported by G12 and Sect. 5.1.2 describes two models which can be implemented within SCIP.

5.1.1 A formulation for the Zinc-modeling language

For the modeling of the starting time of job i and the mode assignment of job i, we use the integer variables s_i and x_i, respectively. With the latter variables and the notation of Sect. 2.1, the MRCPSP with SPRs can be formulated as follows in the

CP-modeling language Zinc:

$$\begin{aligned}
\min \quad & s_{n+1} && (5.1)\\
s.t. \quad & s_i + d_{i,x_i} \leq s_j, \quad \forall j \in \mathfrak{S}_i, \forall i \in J && (5.2)\\
& \sum_{i \in J} c^{\nu}_{i,x_i,r} \leq C^{\nu}_r, \quad \forall r \in N && (5.3)\\
& \texttt{cumulative}(\mathbf{s}, \mathbf{d}, \mathbf{c}^r, C^{\rho}_r), \quad \forall r \in R && (5.4)\\
& s_j \in \{lb(s_j), \ldots, ub(s_j)\}, \quad \forall j \in J && (5.5)\\
& x_j \in M_j, \quad \forall j \in J && (5.6)
\end{aligned}$$

where

$$\begin{aligned}
\mathbf{s} &= [s_j : j \in J] && (5.7)\\
\mathbf{d} &= [d_{j,x_j} : \ j \in J] && (5.8)\\
\mathbf{c}^r &= [c^{\rho}_{j,x_j,r} : \ j \in J] && (5.9)
\end{aligned}$$

The dummy job $n+1$ represents the end of the project, i.e. minimizing the makespan is equal to minimizing s_{n+1} (see 2.1.3 for the detailed characteristics of a dummy job). Furthermore, as a preprocessing step we can remove special mode combinations and deduce the lower and upper bounds $lb(s_j)$ and $ub(s_j)$ for the starting time variables in (5.5 (see Sect. 2.4).

(5.2) are multi-mode precedence constraints. With (5.3) and (5.4), we assure that the available capacities of the nonrenewable and renewable resources are not exceeded. Hereby, in (5.4) we use the scheduling specific global constraint `cumulative` (see also Sect. 3.1.2)[1].

To apply this constraint for the MRCPSP with SPRs, we introduce the variable vectors $\mathbf{s}$, $\mathbf{d}$ and $\mathbf{c}^r$ in (5.7) - (5.9). In the above formulation, variables appear in the indices of parameters, like e.g. in d_{i,x_i}. This modeling technique can only be applied if the respective solver supports the global `element`-constraint introduced by Van Hentenryck and Carillon [85].

In general, the `element`-constraint has the following form (see also Example 3.1.1 in Sect. 3.1.1):

$$\texttt{element}(y, \mathbf{x}, z) \qquad (5.10)$$

(5.10) guarantees, that the y-th element of the variable (or parameter) vector $\mathbf{x}$ equals the variable z, i.e. $\mathbf{x}_y = z$. Clearly, if $\mathbf{x}$ has n entries, it must hold that $y \leq n-1$ if zero is the first index. Propagation algorithms captured by the `element`-constraint can infer domain updates for the variable z in case of domain updates of y or of the entries $\mathbf{x}_i$ of $\mathbf{x}$ and vice versa.

[1] Details regarding the implementation of this constraint in G12 can be found in Schutt [73].

In the case of our model, the terms d_{i,x_i} and $c^{\rho}_{i,x_i,r}$ are internally transformed to new variables d'_i, $c'^{\rho}_{i,r}$ and $c'^{\nu}_{i,r}$ by posting the following constraints:

$$\texttt{element}(x_i, [d_{i,k} : \ k \in M_i], d'_i) \tag{5.11}$$

$$\texttt{element}(x_i, [c^{\rho}_{i,k,r} : \ k \in M_i], c'^{\rho}_{i,r}), \quad \forall r \in R \tag{5.12}$$

$$\texttt{element}(x_i, [c^{\nu}_{i,k,r} : \ k \in M_i], c'^{\nu}_{i,r}), \quad \forall r \in N \tag{5.13}$$

Thus, in our application, after a transformation of the respective solver only the variables d'_i and $c'^{\rho}_{i,r}$ are used in the `cumulative`-constraint.

5.1.2 Formulations for SCIP

SCIP provides the `optcumulative`-constraint introduced by Heinz et al. [42] to model renewable resource constraints in the context of multi-mode jobs. However, to apply the above constraint for the MRCPSP, we have to introduce integer starting time variables $s_{i,k}$ for every job i and mode k and the binary variables $x_{i,k}$ for the mode assignment of job i. Note, that the mode-assignment is modeled by binary variables as SCIP does not support the `element`-constraint.

With the latter variables and the notation of Sect. 2.1, the MRCPSP can be formulated as follows in SCIP with the `optcumulative`-constraint:

$$\min \quad s_{n+1,1} \tag{5.14}$$

$$s.t. \sum_{k \in M_i} x_{i,k} = 1, \quad \forall i \in J \tag{5.15}$$

$$s_{i,k} + d_{i,k} \cdot x_{i,k} \leq s_{j,l}, \quad \forall j \in \mathfrak{S}_i, \forall i \in J, \ \forall k \in M_i, \ \forall l \in M_j \tag{5.16}$$

$$\sum_{i \in J} \sum_{k \in M_i} c^{\nu}_{i,k,r} \cdot x_{i,k} \leq C^{\nu}_r, \quad \forall r \in N \tag{5.17}$$

$$\texttt{optcumulative}(\bar{\mathbf{s}}, \bar{\mathbf{x}}, \bar{\mathbf{d}}, \bar{\mathbf{c}}^r, C^{\rho}_r), \quad \forall r \in R \tag{5.18}$$

$$s_{j,k} \in \{lb(s_{j,k}), \ldots, ub(s_{j,k})\}, \ \forall j \in J, \ k \in M_j \tag{5.19}$$

$$x_{j,k} \in \{0, 1\}, \quad \forall j \in J, \ k \in M_j \tag{5.20}$$

where

$$\bar{\mathbf{s}} = \mathbf{s}^0 \circ \cdots \circ \mathbf{s}^{n+1}, \quad \text{where } \mathbf{s}^i_k = s_{i,k}, \ \forall k \in M_i, \ i \in J \tag{5.21}$$

$$\bar{\mathbf{x}} = \mathbf{x}^0 \circ \cdots \circ \mathbf{x}^{n+1}, \quad \text{where } \mathbf{x}^i_k = x_{i,k}, \ \forall k \in M_i, \ i \in J \tag{5.22}$$

$$\bar{\mathbf{d}} = \mathbf{d}^0 \circ \cdots \circ \mathbf{d}^{n+1}, \quad \text{where } \mathbf{d}^i_k = d_{i,k}, \ \forall k \in M_i, \ i \in J \tag{5.23}$$

$$\bar{\mathbf{c}}^r = \mathbf{c}^{0,r} \circ \cdots \circ \mathbf{c}^{n+1,r}, \text{where } \mathbf{c}^{i,r}_k = c^{\rho}_{i,k,r}, \forall k \in M_i, \ i \in J, r \in R \tag{5.24}$$

To guarantee a correct input for `optcumulative` we have to use the variable vectors $\bar{\mathbf{s}}$ and $\bar{\mathbf{x}}$ and the parameter vectors $\bar{\mathbf{d}}$ and $\bar{\mathbf{c}}^r$ which are given in (5.21)-(5.24).

In this context, the operator $\circ$ is defined as the concatenation of two vectors, whereas the vector $\mathbf{c} = \mathbf{a} \circ \mathbf{b}$ is obtained by appending the elements of $\mathbf{b}$ coordinate-wise to $\mathbf{a}$.

Again, we minimize the starting time $s_{n+1,1}$ of the dummy job $n+1$, which can only be processed in mode 1. With (5.15), (5.16) and (5.17), we formulate the uniqueness of the mode-assignments, the multi-mode precedence constraints and the nonrenewable resource constraints, respectively. (5.18) guarantees that the maximal capacities of the renewable resources are not exceeded.

The above SCIP-formulation has two major disadvantages.

Firstly, we have to introduce starting time variables for every job-mode combination $(i,k),\ i \in J,\ k \in M_i$.

The second disadvantage has to do with the implementation of the `optcumulative`-constraint (see Heinz et al. [42]). The domain propagation step and the inconsistency check for a variable $s_{i,m}$ in the `optcumulative`-constraint only considers variables $s_{j,k}, j \neq i$ for which $x_{j,k} = 1$ in the recent node of the BaB-tree. However, also variables $s_{j,k}, j \neq i$, where the mode-assignment has not been done yet, can be considered for the domain propagation and the inconsistency check of a variable $s_{i,m}$.

To overcome the above disadvantages, we aimed at implementing a new global constraint `cumulativemm` for SCIP to be able to apply a more general form of domain propagation and explanation generation for renewable resources in the context of multi-mode jobs where we only have to introduce starting time variables s_j for every job $j \in J$. The principles of the `cumulativemm`-constraint are outlined in Sect. 5.2.

With our new constraint and again binary variables $x_{i,k}$ for the mode-assignment, we formulate the MRCPSP with SPRs as follows in SCIP:

$$\min \quad s_{n+1} \tag{5.25}$$

$$s.t. \quad \sum_{k \in M_i} x_{i,k} = 1, \quad \forall i \in J \tag{5.26}$$

$$s_i + \sum_{k \in M_i} d_{i,k} \cdot x_{i,k} \leq s_j, \quad \forall j \in \mathfrak{S}_i, \forall i \in J \tag{5.27}$$

$$\sum_{i \in J} \sum_{k \in M_i} c^{\nu}_{i,k,r} \cdot x_{i,k} \leq C^{\nu}_r, \quad \forall r \in N \tag{5.28}$$

$$\texttt{cumulativemm}(\mathbf{s}, \bar{\mathbf{x}}, \bar{\mathbf{d}}, \bar{\mathbf{c}}^r, C^{\rho}_r), \qquad \forall r \in R \tag{5.29}$$

$$s_j \in \{lb(s_j), \ldots, ub(s_j)\}, \quad \forall j \in J,\ k \in M_j \tag{5.30}$$

$$x_{j,k} \in \{0,1\}, \quad \forall j \in J,\ k \in M_j \tag{5.31}$$

5.2 Principles of the `cumulativemm`-constraint

With our `cumulativemm`-constraint, one can model renewable resource constraints for multi-mode jobs. The main ingredients of the `cumulativemm`-constraint are a

feasibility check and functions for preprocessing (see 2.4), constraint propagation and explanation generation (see Sect. 5.2.1 and 5.2.2) for a certain resource $r \in R$.

Our constraint enforces feasibility w.r.t. the renewable resource $r \in R$:

$$\sum_{j \in J,\, m \in M_j:\, t - d_{j,m} + 1 \leq s_j \leq t \,\wedge\, x_{j,m} = 1} c_{j,m,r} \leq C_r^{\rho}, \quad \forall\, t \in \{1, \ldots, T\}$$

The other constraints in the SCIP-model of the MRCPSP with SPRs are modeled by the SCIP-intern constraint handler for linear constraints and can therefore be handled by the SCIP-intern solution principles.

The constraint propagation procedure consists of a redundancy check and a domain reduction step. We firstly check the multi-mode data for redundancy in the current node of the BaB tree. In concrete, if we assume the maximal mode duration, the maximal resource consumption and the maximal processing interval for every job $j \in J$ in the current node and the underlying schedule is feasible w.r.t. the renewable resource $r \in R$, we can locally remove our constraint from the solution procedure. This is due to the fact, that in the above case, it cannot be violated anymore in the succeeding branches of the BaB tree.

The domain reduction step is mainly based on the calculation of a minimal problem instance (MPI) (see Heilmann [39]) in every processed node of the BaB tree, i.e. the transformation of the multi-mode data to a single-mode representative.

We assume that the uniqueness of the mode-assignments can still be fulfilled in the succeeding nodes of the BaB tree, i.e. it is not the case, that:

$$\begin{aligned} \exists\, i \in J: &\qquad |\{k \in M_i:\ lb(x_{i,k}) = 1\}| \geq 2 \\ or\ \exists\, i \in J: &\qquad \forall k \in M_i:\ ub(x_{i,k}) = 0 \end{aligned}$$

If one of the above cases is present, we cut off the current node of the BaB tree and initialize the SCIP-intern CA and we do not apply our propagation algorithms.

Otherwise, we calculate a minimal processing version $MPV_{j,r} = (\text{domain}(s_j); d_j^{\min}; c_{j,r}^{\min})$ for every job $j \in J$ and renewable resource $r \in R$ as follows:

$$d_j^{\min} = \min_{k \in M_j} \{d_{j,k} : ub(x_{j,k}) > 0\} \tag{5.32}$$

$$c_{j,r}^{\min} = \min_{k \in M_j} \left\{ c_{j,k,r}^{\rho} : ub(x_{j,k}) > 0 \right\} \tag{5.33}$$

In (5.32) and (5.33), we calculate the minimal duration and resource consumption of resource $r \in R$ w.r.t. the modes which have not been excluded ($ub(x_{j,k} > 0$) in the recent node of the BaB tree.

With the MPI at hand, we can apply standard constraint propagation algorithms for renewable resources like e.g. TP and EF (see Sect. 3.1.2). Our current implementation of the `cumulativemm`-constraint only integrates TP.

In case of the MRCPSP, the compulsory part cp_j of a job j (see also Sect. 3.1.2) is defined as follows:

$$cp_j = \begin{cases} \left\{ub(s_j), \ldots, lb(s_j) + d_j^{\min} - 1\right\}, & \text{if } lb(s_j) + d_j^{\min} > ub(s_j) \\ \emptyset, & \text{else} \end{cases} \tag{5.34}$$

Example 5.2.1. ***TP for multi-mode jobs*** *Assume that in the course of the BaB-algorithm of SCIP and after the redundancy check, our constraint propagation procedure has the following input:*

$$(\text{domain}(s_1), \text{domain}(x_{1,1}), d_{1,1}, c^{\rho}_{1,1,1}) = (\{3,4,5\}, \{0,1\}, 2, 1)$$
$$(\text{domain}(s_1), \text{domain}(x_{1,2}), d_{1,2}, c^{\rho}_{1,2,1}) = (\{3,4,5\}, \{0,1\}, 3, 2)$$
$$(\text{domain}(s_2), \text{domain}(x_{2,1}), d_{2,1}, c^{\rho}_{2,1,1}) = (\{2,3,4\}, \{1\}, 3, 2)$$

The maximal capacity of the renewable resource 1, $C^{\rho}_1 = 2$. We can see that job 2 is processed in mode 1, as $x_{2,1} = 1$. Thus, $d_2^{min} = 3$ and $c_{2,1}^{min} = 2$. Moreover, job 2 is surely processed at the time point 4, as its compulsory part $cp_2 = \{4\}$. Next, we consider job 1 with $d_1^{min} = 2$ and $c_{1,1}^{min} = 1$. $cp_1 = \emptyset$ but we can deduce a domain update. As $lb(s_1) + d_1^{min} = 3 + 2 \geq 4$, starting job 1 at its lower bound would lead to a resource conflict at the time point 4. The TP algorithm will find the largest time point $t_1 - 1 = 4$ such that the capacity is violated $(2 + 1 > 2)$. After that $lb(s_1)$ would be updated to $t_1 = 5$ which equals $ub(s_1)$ and a new compulsory part of job 1 $cp_1 = \{5, 6\}$ is evaluated.

Note, that the principles of our constraint propagation procedure are standard techniques. These are applied in a similar way in CP solvers like e.g. JaCoP [46] which provide the `cumulative`-constraint supporting variable durations and resource consumptions for every job.

The idea of integrating explanation generation, i.e. of processing reasons for the deduced domain reductions or inconsistencies to a SAT solving mechanism is rather new. To our knowledge, there are only two optimization frameworks integrating this feature, i.e. SCIP and G12.

Schutt [73] describes principles for explanation generation in the context of jobs having variable durations and resource consumptions. These explanation generation techniques are integrated in the `cumulative`-constraint provided by the G12. In our `cumulativemm`-constraint, we explain the reasons for the domain reductions or inconsistencies in a different way. In the next two sections we introduce our strategy for explanation generation and compare it to the strategy of Schutt [73].

5.2.1 Explanations for TP with multi-mode jobs

In order to integrate our constraint into the SCIP-intern CA mechanism, we have to provide functions for the `cumulativemm`-constraint which derive explanations for the inconsistencies or domain updates detected by the TP algorithm. In addition to the

Boolean literals integrating s_j (see Sect. 3.3.1), the explanations of `cumulativemm` also consist of $[\![x_{j,k} = 0]\!]$.

Assume now, the TP algorithm found an inconsistency because of the jobs having their compulsory parts cp_j (see (5.34)) at time t_κ and cause a resource violation:

$$\sum_{j:t_\kappa \in cp_j} c_{j,r}^{min} > C_r^\rho$$

Then, our constraint handler derives the following explanation:

$$\bigwedge_{j:t_\kappa \in cp_j} E_j \Longrightarrow \texttt{false} \tag{5.35}$$

(5.35) can be divided into the sub-explanations E_j for every job participating in the conflict:

$$\begin{aligned} E_j = \quad & [\![t_\kappa - d_j^{min} + 1 \leq s_j]\!] \wedge [\![s_j \leq t_\kappa]\!] \\ \wedge \quad & \bigwedge_{m:m \in M_j \text{ and } ub(x_{j,m})=0} [\![x_{j,m} = 0]\!] \end{aligned} \tag{5.36}$$

E_j is correct as in a situation, where the variables s_j and $x_{j,m}$ have bounds as in (5.36), the compulsory parts cp_j of the involved jobs would again include the time point t_κ and this would again lead to a resource violation. Note, that d_j^{min} and $c_{j,r}^{min}$ are calculated by (5.32) and (5.33), respectively.

In addition to explanations for inconsistencies, our constraint handler also processes explanations for domain updates deduced by our TP algorithm to the SCIP-intern CA mechanism. SCIP stores information about the time the bound changes took place and about the constraints which processed the domain updates through a so-called bound change index (BCI). In the course of the BaB-algorithm, the SCIP-intern CA can ask our constraint handler for an explanation of the new lower bound $lb^*(s_i)$ of job i at the BCI b, if it was deduced by the `cumulativemm`-constraint.

Therefore, we introduce the time point $t = lb^*(s_i) - 1$. Jobs $j \in J \backslash \{i\}$ with compulsory parts cp_j^b where $t \in cp_j^b$ such that:

$$c_{i,r}^{min,b} + \sum_{j \neq i: t \in cp_j} c_{j,r}^{min,b} > C_r^\rho$$

were responsible for the bound change at the BCI b. Additionally the lower bound $lb(s_i)$ of job i has to exceed a certain value for the domain update. The complete explanation consists of two clauses e_i and f, where e_i contains the minimal lower bound of s_i and f the compulsory parts cp_j^b of jobs $j \neq i$ with $t \in cp_j^b$. e_i is given as follows:

$$e_i = \quad [\![t - d_i^{min,b} + 1 \leq s_i]\!] \quad \wedge \bigwedge_{m:m \in M_i \text{ and } ub(x_{i,m}^b)=0} [\![x_{i,m} = 0]\!]$$

The clause f is as follows:

$$f = \bigwedge_{j:j\neq i \text{ and } t\in cp_j^b} E_j$$

where E_j is given through a small variation of (5.36). We replace t_κ by t, $d_j^{\min}$ by $d_j^{\min,b}$ and $ub(x_{j,m})$ by $ub^b(x_{j,m})$. Thus, we evaluate the latter values for the given BCI b. The complete explanation for the bound change $lb(s_i) \rightarrow lb^*(s_i)$ is given through:

$$e_i \wedge f \Longrightarrow [\![lb^*(s_i) \leq s_i]\!] \tag{5.37}$$

The argument for the correctness of the above explanation is the same as in the inconsistency case. Note, that the explanation (5.37) depends on the BCI b.
In our TP algorithm we process the bound changes in a pointwise manner, i.e. we guarantee that $lb^*(s_i) - lb(s_i) \leq d_j^{min,b}$. With this, we want to imitate the pointwise explanations proposed by Schutt et al. [74]. The explanations for the upper bound changes are processed in a symmetric way.

The following example illustrates a possible outcome of our explanation generation procedure.

Example 5.2.2. *Firstly, we extend Example 5.2.1 by another job with two modes and the following input:*

$$(\text{domain}(s_3), \text{domain}(x_{3,1}), d_{3,1}, c^{\rho}_{3,1,1}) = (\{5\}, \{1\}, 2, 2)$$

Note, that there is a resource conflict at time point $t_\kappa = 6$, as $6 \in cp_1 \cap cp_3$ and $c^{\rho}_{3,1,1}(=2) + c^{min}_{1,1}(=1) > 2$. The explanation for this inconsistency is as follows:

$$([\![5 \leq s_1]\!] \wedge [\![s_1 \leq 6]\!]) \wedge ([\![5 \leq s_3]\!] \wedge [\![s_3 \leq 6]\!] \wedge [\![x_{3,2} = 0]\!]) \Longrightarrow \texttt{false} \tag{5.38}$$

Note, that job 3 is processed in mode 1 and for job 1 it holds, that $ub(x_{1,k}) > 0$, $\forall k \in M_1$. After the initialization of the SCIP-intern CA, SCIP asks our constraint handler for the reason of the lower bound change of s_1 from 3 to 5 from Example 5.2.1, i.e. an explanation for the literal $[\![5 \leq s_1]\!]$.

Our constraint handler gives the following explanation:

$$[\![3 \leq s_1]\!] \wedge ([\![3 \leq s_2]\!] \wedge [\![s_2 \leq 4]\!] \wedge [\![x_{2,2} = 0]\!]) \Longrightarrow [\![5 \leq s_1]\!] \tag{5.39}$$

Every boolean literal from (5.38) and (5.39) is added as a new node to the SCIP-intern conflict graph. Moreover, an arc is constructed from every boolean literal of the left-hand side of the explanation to the boolean literal on the right-hand side.

5.2.2 Comparison to other explanation generation techniques and possible improvements

Schutt [73, p.96] also introduces explanations for the `cumulative`-constraint where the durations and the resource consumptions of the jobs can be variables. In our

G12-model of Sect. 5.1, we use this constraint with the duration vector $\mathbf{d}$ and the resource consumption vector $\mathbf{c}_r^{\rho}$ to model the resource constraint for the renewable resource $r \in R$. After the transformation given by (5.11) and (5.12), the G12 solution approach will only use the variable vectors $\mathbf{d}'$ and $\mathbf{c}_r'^{\rho}$ in the `cumulative`-constraint. These are connected to the original durations and resource consumptions by the `element`-constraint (see (5.11) and (5.12)).

With our notation, the preliminary version of the explanations for a lower bound update of s_i to $lb^*(s_i)$ applied in the `cumulative`-constraint (see Schutt [73, p.96]) are as follows (at the BCI b):

$$\begin{aligned} & [\![lb^*(s_i) - lb^b(d'_i) \le s_i]\!] \wedge [\![lb^b(d'_i) \le d'_i \,]\!] \wedge [\![lb^b(c'^{\rho}_{i,r}) \le c'^{\rho}_{i,r} \,]\!] \quad \wedge \\ \bigwedge_{j:j\neq i \text{ and } t\in cp_j^b} & [\![lb^*(s_i) - lb^b(d'_j) \le s_j]\!] \wedge [\![s_j \le lb^*(s_i) - 1]\!] \wedge [\![lb^b(d'_j) \le d'_j \,]\!] \quad \wedge \\ \bigwedge_{j:j\neq i \text{ and } t\in cp_j^b} & [\![lb^b(c'^{\rho}_{j,r}) \le c'^{\rho}_{j,r} \,]\!] \quad \Longrightarrow \quad [\![lb^*(s_i) \le s_i]\!] \end{aligned} \tag{5.40}$$

Schutt [73] notes, that these explanations can be strengthened by choosing different values q_i and l_i instead of $lb^b(d'_i)$ and $lb^b(c'^{\rho}_{i,r})$, respectively. For example consider the case where the domain of d'_i is internally encoded as a range of consecutive values but the set $\mathcal{D}_i = \{d_{i,m},\ m \in M_i\}$ consists of nonconsecutive values and it holds that $lb^b(d'_i) \notin \mathcal{D}_i$. Then, by using $q = \min\{d_{i,m} \ :\ d_{i,m} \ge lb^b(d'_i)\}$, the explanation (5.40) can be strengthened. Schutt [73] specifies the values q_i and l_i which lead to the strongest explanations.

In our explanation for a domain update (see (5.37)), we omit the part where the resource consumptions of the involved jobs are explained as in (5.40). This is due to the fact, that as soon as certain modes are excluded, we can reason about the minimal duration $d_j^{\min}$ and the minimal resource consumption $c_{j,r}^{\min}$. Thus we do not have to introduce explanations for both durations and resource consumptions. This can be an advantage compared to the explanations of Schutt [73].

Assume therefore, that we are in a situation in a node c of the BaB tree where the input is the same for both algorithms, and both algorithms already generated the explanations (5.37) and (5.40), respectively at node b with the same set of jobs J_{exp} involved in both explanations. Additionally,

$$d_j^{\min,b} = lb^b(d'_j),\ \forall j \in J_{\text{exp}} \tag{5.41}$$

$$c_{j,r}^{\min,b} = lb^b(c'^{\rho}_{j,r}),\ \forall j \in J_{\text{exp}} \tag{5.42}$$

Moreover, the left hand side of (5.37) is true at node c and

$$\exists\, k \in J_{\text{exp}} :\ \ t_b \in cp_k^c \wedge lb^b(c'^{\rho}_{k,r}) > lb^c(c'^{\rho}_{k,r}) \tag{5.43}$$

whereat the first two lines of (5.40) are true. As (5.37) is true, our algorithm will immediately deduce $lb^*(s_i) \le s_i$.

Because of (5.43), the G12-algorithm will not immediately deduce the latter lower bound update. As the first two lines of (5.40) are true and (5.41) and (5.42) hold, we can conclude that

$$x_j \in M_j^b = \{m : m \in M_j \text{ and } ub(x_{j,m}^b) > 0\},\ \forall j \in J_{\text{exp}} \tag{5.44}$$

and thus $lb^b(c'^{\rho}_{k,r}) \leq c'^{\rho}_{k,r}$, i.e. $lb^c(c'^{\rho}_{k,r})$ can be updated to $lb^b(c'^{\rho}_{k,r})$. Now, the complete left hand side of (5.40) is true and the G12-algorithm will also deduce $lb^*(s_i) \leq s_i$.

The update of $lb^c(x_k)$ outlined in (5.44) and the update of $lb^c(c'^{\rho}_{k,r})$ have to be processed by the `element`-constraints (5.11) and (5.12) in the G12-algorithm before the explanation (5.40) leads to the update of $lb^*(s_i)$. As this update happens immediately with our explanation (5.37), there are cases where our explanation generation strategy can lead to time savings.

Our explanations can also be strengthened.

Example 5.2.3. *Consider a job 1 with the following input at the current node:*

$$\begin{aligned}(\text{domain}(x_{1,1}), d_{1,1}, c^{\rho}_{1,1,1}) &= (\{0\}, 2, 2)\\ (\text{domain}(x_{1,2}), d_{1,2}, c^{\rho}_{1,2,1}) &= (\{0\}, 3, 3)\\ (\text{domain}(x_{1,3}), d_{1,3}, c^{\rho}_{1,3,1}) &= (\{1\}, 4, 3)\end{aligned}$$

Assume that our TP algorithm would detect an inconsistency at the time point 4 *and job 1 is involved in the latter inconsistency, i.e.* $4 \in cp_1$. *As* $d_1^{\min} = 4$, *the part of the explanation containing job 1 is as follows:*

$$[\![1 \leq s_1]\!] \wedge [\![s_1 \leq 4]\!] \wedge [\![x_{1,1} = 0]\!] \wedge [\![x_{1,2} = 0]\!]$$

If the global domain of s_1 *equals* $\{2, \ldots, 6\}$, *we can strengthen the explanation as*

$$\begin{aligned}&[\![1 \leq s_1]\!] \text{ is globally true},\\ &[\![x_{1,1} = 0]\!] \wedge [\![x_{1,2} = 0]\!] &&\Longrightarrow [\![x_{1,1} = 0]\!],\\ &[\![x_{1,1} = 0]\!] \wedge [\![s_1 \leq 4]\!] &&\Longrightarrow 4 \in cp_1,\\ &c^{\rho}_{1,2,1} = c^{\rho}_{1,3,1}.\end{aligned}$$

Thus, we can use the following stronger explanation for the compulsory part of job 1:

$$[\![s_1 \leq 4]\!] \wedge [\![x_{1,1} = 0]\!]$$

Motivated by the above example, assume that job i is part of the job set J_{exp} involved in the explanation (5.37) w.r.t. time point t. Moreover, let $lb^g(s_i)$ be its global lower bound. We can possibly strengthen the explanation (5.37) by strengthening the part of the explanation integrating job $i \in J_{\text{exp}}$.

This can done in two steps:
Firstly, if $t - d_i^{\min} + 1 < lb^g(s_i)$, omit $[\![t - d_i^{\min} + 1 \leq s_i]\!]$.

Set $d_i^{\min} :\leftarrow \min\{d_i^{\min}, t - lb^g(s_i) + 1\}$.
Secondly, we determine the set B_i^* consisting of the modes $m \in M_i$ which fulfill:

$$d_{i,m} \geq d_i^{\min} \tag{5.45}$$

$$c_{i,m,r}^{\rho} + \sum_{j \in J_{\exp} \setminus \{i\}} c_{j,r}^{\min} > C_r^{\rho} \tag{5.46}$$

Now, we can substitute

$$\bigwedge_{m:m \in M_i \text{ and } ub(x_{i,m}^b)=0} [\![x_{i,m} = 0]\!]$$

by

$$\bigwedge_{m:m \in M_i \setminus B_i^*} [\![x_{i,m} = 0]\!]$$

Because of the evaluation of $d_i^{\min}$ and $c_{i,r}^{\min}$ in (5.32) and (5.33) and because of (5.45) and (5.46) it holds that

$$M_i \setminus B_i^* \subseteq \{m : m \in M_i \text{ and } ub(x_{i,m}) = 0\}$$

Thus, the part of the explanation (5.37) integrating job i is possibly stronger. Finally, we update $c_{i,r}^{\min} :\leftarrow \min\{c_{i,m,r}^{\rho} : m \in B_i^*\}$. After the latter update, we continue with the next non-processed job.

5.3 Computational experiments

The three CP models from Sect. 5.1 were solved on the Vienna Scientific Cluster (VSC). Details about the system architecture can be found in Sect. 4.3.

For the solution of the models from Sect. 5.1.1, we used the G12 Constraint Programming Platform [31] 2.0.0 provided by the NICTA research team [60]. Thereby, we formulated the models in Zinc and solved them by the LCG-plugin `g12_fdx`. For the implementation of the constraint handler `cumulativemm` and the formulation and solution of the SCIP-models of Sect. 5.1.2, we used SCIP 3.1.0 in combination with the programming languages `C/C++`. We set the parameters in SCIP such that feasibility is detected fast (with `SCIP_PARAMEMPHASIS_FEASIBILITY`). Furthermore, we impose a memory limit of 2GB RAM for instances with less than 100 jobs and of 3GB for the 100-job instances.

The three CP-models are denoted by the following abbreviations:

`G12` The model from Sect. 5.1.1 formulated in Zinc.

`SCIPopt` The SCIP-model of Sect. 5.1.2 integrating the existing `optcumulative`-constraint.

`SCIP` The SCIP-model of Sect. 5.1.2 integrating our `cumulativemm`-constraint.

Moreover, for every model we distinguish two solution approaches which differ in the generation of the initial domains:

`Max` The initial domains of the starting time variables are evaluated by forward (backward) recursion based on the upper bound $T_{\max}$ given in (2.10).

`Best` The initial domains are generated based on twelve different upper bounds $T_1, \ldots, T_{12}$ where T_1 equals the best known upper bound from the literature and $T_l = T_{l-1}+4$, $\forall l = 2, \ldots, 12$. A model is run on the processor $l = 1, \ldots, 12$ with initial domains based on T_l. Hence, in this case we apply a parallel approach which applies twelve processors. In the end, we take the best results w.r.t. all twelve processors.

Preliminary experiments have shown that like in Sect. 4 varying the initial domains based on different upper bounds on the makespan also leads to a performance variability of the CP-SAT approaches for the MRCPSP with SPRs. With the parallel approach, we want to take advantage of this performance variability as proposed in Sect. 4.

All in all, we compare six different solution approaches to the state-of-the-art exact approach (Branch-and-Cut) of Zhu et al. [92] for the MRCPSP with SPRs (`MMBAC`). All approaches are tested on the 554 and 552 feasible instances with 20 and 30 jobs from the PSPLIB (Kolisch and Sprecher [50]) (`J20mm` and `J30mm`). Moreover, we evaluate the approaches which apply our constraint handler (`SCIPMax` and `SCIPBest`) on the new 540 MRCPSP-instances with 50 and 100 jobs (`MMLIB50` and `MMLIB100`) provided by the Operations Research and Scheduling research group of the University of Ghent [64]. For instances with 20, 30, 50 and 100 jobs, we abort the solution process after a time limit of 1200s, 2400s, 5400s and 7200s, respectively. To assure a fair comparison to the approach of Zhu et al. [92], we used time limits which are 2/3 of their time limits for the 20- and 30-job instances, as their processor is ≈ 1.5 times slower than ours.

We compare the different approaches based on the following measurements:

`#feas` Number of instances where a feasible solution could be found within the given time limit.

`#opt` Number of instances solved to optimality within the given time limit.

`#best` Number of solutions whose makespan equals or improves the best known makespan from the literature.

t_{tot} Average solution times for all instances (we take the minimal solution times of all processors for one instance in the parallel approach).

$G_{opt}(\%)$ The average optimality gap evaluated by SCIP for the instances where a feasible solution was found (we take the optimality gap of the processor which found the best makespan in the parallel approach).

`#Imp` Number of instances where we could improve the best known makespan from the literature.

Table 5.1 shows the results for our models and the state-of-the-art exact approach of Zhu et al. [92] (`MMBAC`) on the `J20mm` instances. For these instances `MMBAC` outperforms all of our single-core approaches (`G12Max`, `SCIPMax` and `SCIPoptMax`) as they can solve all feasible instances to optimality. Nevertheless, `G12Max` is highly competitive to the approach of Zhu et al. [92]. The parallel approach `G12Best` outperforms the state-of-the-art exact approach from the literature. We can also solve all feasible instances to optimality, but `MMBAC` is three times slower.

Furthermore, the SCIP-approaches using our `cumulativemm`-constraint significantly outperform the SCIP-approaches integrating the existing `optcumulative`-constraint. With `SCIPBest` we can solve 40 more instances to optimality and are on average approximately five times as fast as `SCIPoptBest`. Moreover, `SCIPBest` is competitive to `MMBAC` and `G12Best`.

	`#feas`	`#opt`	`#best`	t_{tot}	$G_{opt}(\%)$	`#Imp`
`G12Max`	554	552	554	20.58	0.05	0
`G12Best`	554	554	554	10.39	0.0	0
`SCIPMax`	554	547	550	33.96	0.32	0
`SCIPBest`	554	552	554	22.74	0.07	0
`SCIPoptMax`	554	504	519	130.37	3.16	0
`SCIPoptBest`	554	512	543	113.66	2.85	0
`MMBAC`	554	554	554	32.06	0	-

Table 5.1: Results on the 20-job instances

Table 5.2 shows the results for the `J30mm` instances. Here, we can already solve 9 more instances to optimality than `MMBAC` with the single-core approach `G12Max`. Moreover, `G12Max` is almost two times faster than `MMBAC`. The parallel approach `G12Best` significantly outperforms `MMBAC` both regarding average solution times and solution quality. Again, `SCIPBest` (`SCIPMax`) is considerably better than `SCIPoptBest` (`SCIPoptMax`). We can solve 36 (44) more instances to optimality and are $\approx 42\%$ (43%) faster.

As the Branch-and-Cut approach of Zhu et al. [92] is based on a MIP formulation of the MRCPSP, it is highly dependent of a starting solution with a small makespan to reduce the number of binary variables. They use starting solutions computed by a problem specific heuristic whose makespan on average only deviates by 2.18% from the best known makespans of the PSPLIB. An advantage of our SCIP- and G12 approaches is that they still produce good results with the relatively high upper bound $T_{\max}$ as input.

	#feas	#opt	#best	t_{tot}	$G_{opt}(\%)$	#Imp
G12Max	552	515	521	212.44	2.59	0
G12Best	552	521	537	180.5	1.97	0
SCIPMax	552	500	508	259.71	3.37	0
SCIPBest	552	504	517	232.95	2.96	0
SCIPoptMax	552	456	470	452.46	8.15	0
SCIPoptBest	552	468	480	401.74	7.29	0
MMBAC	552	506	529	393.13	-	-

Table 5.2: Results on the 30-job instances

Tables 5.3 and 5.4 contain the results for the runs with the 50- and 100-job instances, respectively. To our knowledge, these have not been solved exactly before. For these instance sets, the approaches integrating the best SCIP model significantly outperform the approaches integrating the G12-model.

We can solve approx. 78% of the 50-job instances and 63% of the 100-job instances to optimality within the given time limits with SCIPBest. Moreover, we improved the best known makespans of 70 instances with 50 jobs and 106 instances with 100 jobs. These makespans were reported by the Operations Research and Scheduling research group of the University of Ghent [64] and were evaluated in the course of the experiments described by Van Peteghem and Vanhoucke [87].

	#feas	#opt	#best	t_{tot}	$G_{opt}(\%)$	#Imp
SCIPMax	540	405	422	1409.69	17.73	59
SCIPBest	540	420	445	1252.25	9.86	70
G12Max	532	363	383	1952.36	11.04	61
G12Best	539	367	420	1861.9	8.86	68

Table 5.3: Results on the 50-job instances

	#feas	#opt	#best	t_{tot}	$G_{opt}(\%)$	#Imp
SCIPMax	518	312	328	3127.93	327.42	98
SCIPBest	535	338	353	2740.31	33.4	106
G12Max	219	150	160	5963.62	8.77	25
G12Best	404	245	270	5255.9	12.82	52

Table 5.4: Results on the 100-job instances

5.4 Conclusion

In this chapter, we introduced a generalization of the exact CP-SAT approaches for the SRCPSP with SPRs to the MRCPSP with SPRs. This generalization can on

the one hand be achieved on the modeling level. We introduced formulations of the MRCPSP with SPRs in optimization frameworks (G12 and SCIP) which integrate a BaB-algorithm in combination with CP and SAT techniques. One formulation is usable via the Zinc modeling language which supports the solution of models by LCG, the state-of-the-art exact approach for variants of the SRCPSP with SPRs. Moreover, we proposed two formulations for the optimization framework SCIP. One of the latter is based on our new constraint handler called `cumulativemm` and the other one on the existing `optcumulative`-constraint introduced by Heinz et al. [42].

The computational experiments show that for the 20- and 30-job instances at least one of our proposed approaches can outperform the state-of-the-art exact algorithm of Zhu et al. [92] for the MRCPSP with SPRs. Our parallel G12-algorithm is almost three times faster than their approach on the 20-job instances. Moreover, on the 30-job instances already our single-core approach using a trivial upper bound as input for the G12-algorithm can solve 9 more instances to optimality and is almost two times faster. A clear advantage of our approaches is that they also produce good results when large upper bounds are used as input. In contrast, the quality of the approach of Zhu et al. [92] is highly dependent on small UBs as input to reduce the initial number of variables.

Moreover, the SCIP-approaches which apply our `cumulativemm`-constraint are significantly better than the SCIP-approaches integrating the existing `optcumulative`-constraint. For the 20- and 30-job instances, we could solve 40 and 36 more problem instances to optimality in approximately five and two times smaller average solution times, respectively.

Furthermore, we are the first to exactly solve new instances of the MRCPSP with SPRs with 50 and 100 jobs from the literature. In contrast to the 20- and 30-job instances, the SCIP approaches integrating `cumulativemm` are significantly better than the G12 solution approaches on these instance sets. The best performing SCIP approach solves approx. 78% of the 50-jobs instances and 63% of the 100-jobs instances to optimality within the time limit of 5400s and 7200s, respectively. Moreover, we could improve the best known makespans evaluated by the heuristic approaches presented by Van Peteghem and Vanhoucke [87] for 70 and 106 instances with 50 and 100 jobs, respectively.

Chapter 6

CP-SAT approaches for the MRCPSP with GPRs

6.1 Introduction

The following chapter proposes extensions of the CP-SAT approaches from Chapt. 5 to solve the MRCPSP with GPRs. In Sect. 6.2, we formulate the MRCPSP with GPRs in SCIP and show how our model can be strengthened by the integration of linear constraints, `sprecedencemm` and `gprecedencemm`. `sprecedencemm` and `gprecedencemm` are two new constraint handler which integrate problem-specific propagation and explanation generation algorithms based on SPRs and GPRs in the context of multi-mode activities. In Sect. 6.3 and 6.4, we present the key principles of our new global constraints `sprecedencemm` and `gprecedencemm`, respectively. In Sect. 6.5 the different SCIP-models are computationally evaluated and the chapter ends with a conclusion on the obtained results in Sect. 6.6.

6.2 A SCIP-model for the MRCPSP with GPRs

For the modeling of the starting time of job i and the mode assignment of job i, we use the integer variables s_i and the binary variables $x_{i,k}$, respectively. With the

notation of Sect. 2.1, the MRCPSP with GPRs can be formulated as follows in SCIP:

$$\min \quad s_{n+1} \tag{6.1}$$

$$s.t. \sum_{k \in M_i} x_{i,k} = 1, \quad \forall i \in J \tag{6.2}$$

$$\sum_{i \in J} \sum_{k \in M_i} c^{\nu}_{i,k,r} \cdot x_{i,k} \leq C^{\nu}_r, \quad \forall r \in N \tag{6.3}$$

$$-(1 - x_{i,k}) \cdot M + s_i + \delta_{i,j,k,l} \leq s_j + (1 - x_{j,l}) \cdot M,$$
$$\forall j \in \mathfrak{S}_i, \forall i \in J, \ \forall k \in M_i, \ \forall l \in M_j \tag{6.4}$$

$$\texttt{cumulativemm}(\mathbf{s}, \mathbf{x}, \mathbf{d}, \mathbf{c}^r, C^{\rho}_r), \quad \forall r \in R \tag{6.5}$$

$$s_j \in \{lb(s_j), \ldots, ub(s_j)\}, \forall j \in J \tag{6.6}$$

$$x_{j,k} \in \{0, 1\}, \quad \forall j \in J, \ k \in M_j \tag{6.7}$$

where

$$\mathbf{s} = \bigcirc_{i \in J} \mathbf{s}^i, \quad \text{where } \mathbf{s}^i = [s_i], \ i \in J \tag{6.8}$$

$$\mathbf{x} = \bigcirc_{i \in J} \mathbf{x}^i, \quad \text{where } \mathbf{x}^i_k = x_{i,k}, \ \forall k \in M_i, \ i \in J \tag{6.9}$$

$$\mathbf{d} = \bigcirc_{i \in J} \mathbf{d}^i, \quad \text{where } \mathbf{d}^i_k = d_{i,k}, \ \forall k \in M_i, \ i \in J \tag{6.10}$$

$$\mathbf{c}^r = \bigcirc_{i \in J} \mathbf{c}^{i,r}, \text{where } \mathbf{c}^{i,r}_k = c^{\rho}_{i,k,r}, \forall k \in M_i, \ i \in J, r \in R \tag{6.11}$$

In the above model, we use the variable vectors $\mathbf{s}$ and $\mathbf{x}$ and the parameter vectors $\mathbf{d}$ and $\mathbf{c}^r$ which are given in (6.8)-(6.11). In this context, the operator $\circ$ is defined as in Sect. 5.1.2.

With (6.2) and (6.3), we assure the uniqueness of the mode-assignments and formulate the nonrenewable resource constraints, respectively. With (6.4), we model GPRs between multi-mode jobs. With a sufficiently big constant M, the constraint only becomes binding for the variables s_i and s_j, if $x_{i,k} = 1$ and $x_{j,l} = 1$.

In (6.5), we apply `cumulativemm` as in Sect. 5.1.2 to model renewable resource constraints for multi-mode jobs.

To reduce the initial search space for the solution algorithm, again lower bounds $lb(s_i)$ and upper bounds $ub(s_i)$ are computed for s_i as proposed in Sect. 2.4.

6.2.1 Strengthening of the mathematical formulation

In the following, we propose three variants for the strengthening of the above formulation of the MRCPSP with GPRs. Firstly, the above model can be strengthened by the addition of the linear constraints:

$$s_i + \sum_{k \in M_i} d^*_{i,k} \cdot x_{i,k} \leq s_j, \quad \forall j \in \mathfrak{S}_i, \forall i \in J \tag{6.12}$$

$$\text{where} \qquad d^*_{i,k} = \min_{j \in S_i, l \in M_j} \{\delta_{i,j,k,l}\} \tag{6.13}$$

The model (5.14) - (5.20) is equivalent to the model (5.14) - (5.20) with (6.12). However, the latter model has a potentially stronger LP-relaxation and leads to an improvement of the propagation algorithms w.r.t. the variables s_i in the course of the solution algorithm.

For the second and third strengthening variant, we implemented two new constraint handlers for SCIP, namely `sprecedencemm` and `gprecedencemm`, which can be used for the modeling of SPRs and GPRs in the context of multi-mode activities, respectively.

One can model SPRs as follows:

$$\texttt{sprecedencemm}(\mathbf{s}, \mathbf{x}, \mathbf{d}, \mathfrak{S}) \tag{6.14}$$

Thereby, the input is defined by (5.7) - (6.10) and the vector $\mathfrak{S}$ which contains the successor sets $\mathfrak{S}_i$ for every activity i.

`sprecedencemm` guarantees feasibility w.r.t. the following formula:

$$\text{If } x_{i,k} = 1 \Rightarrow s_i + d_{i,k} \leq s_j, \quad \forall j \in \mathfrak{S}_i, \forall i \in J, \forall k \in M_i \tag{6.15}$$

Moreover, it also captures propagation and explanation generation algorithms for SPRs in the context of multi-mode activities (see Sect. 6.3).

`sprecedencemm` can also be applied in the context of GPRs to strengthen the model (5.14) - (5.20). Therefore, we substitute the vector $\mathbf{d}$ in (6.14) by

$$\mathbf{d}^* = \bigcirc_{i \in J} \mathbf{d}_i^* \tag{6.16}$$

where $\mathbf{d}_i^*$ contains the values $d_{i,k}^*$, $\forall k \in M_i$ defined in (6.13). In other words, we add the constraint

$$\texttt{sprecedencemm}(\mathbf{s}, \mathbf{x}, \mathbf{d}^*, \mathfrak{S}) \tag{6.17}$$

to the model (5.14) - (5.20).

Example 6.2.1. *For an illustration of the input needed for the constraint (6.17), we stick to the data introduced in the example of Sect. 2.5. We restrict ourselves to the data that has to be collected w.r.t. the precedence relation* $(1, 4)$.

For this strengthening variant, we have to determine the duration vector $\mathbf{d}^*$ *for* `sprecedencemm`.

Therefore, we firstly determine the minimal time lags $d_{1,k}^*$, $k \in \{1, 2\}$ *w.r.t. all modes of the successor 4:*

$$d_{1,1}^* = -5, \; d_{1,2}^* = -6,$$

i.e. we obtain the vector $d_1^* = [-5, -6]$.

The computation of $d_{j,k}^*$, $k \in \{1, 2\}$ *for the remaining jobs* $j \in \{0, 2, 3, 4, 5, 6, 7\}$ *is similar.*

The vectors $\mathbf{d}_i^*$, $i \in \{0, \ldots, 7\}$ *are then concatenated to the vector* $\mathbf{d}^*$ *and used as input for* `sprecedencemm`. *Note, that if an activity* j *would have more than one successor, we would have to determine* $d_{j,k}^*$, $k \in \{1, 2\}$ *w.r.t. all successors and all the respective modes. The generation of the vectors* $\mathbf{s}$, $\mathbf{x}$ *and* $\mathfrak{S}$ *is straightforward.*

Again, the formulation (5.14) - (5.20) is equivalent to the formulation (5.14) - (5.20) with (6.17). However, as `sprecedencemm` captures problem-specific propagation and explanation generation algorithms, the addition of (6.17) potentially enhances the overall solution procedure applied by SCIP.

The second strengthening variant with (6.17) has an advantage over the first strengthening variant with (6.12). In contrast to the constraint handler of SCIP which manages linear constraints [1], and thus (6.12), the propagation and explanation generation algorithms of `sprecedencemm` integrate the knowledge that exactly one mode has to be chosen for every activity (see Sect. 6.3). This potentially leads to stronger domain reductions.

GPRs can be modeled by `gprecedencemm` in the following way:

$$\texttt{gprecedencemm}(\mathbf{s}, \mathbf{x}, \boldsymbol{\delta}, \mathfrak{S}), \tag{6.18}$$

$$\begin{aligned} &\text{where} && \boldsymbol{\delta} = \bigcirc_{i \in J, j \in \mathfrak{S}_i} \boldsymbol{\delta}^{i,j}, && (6.19)\\ &\text{and} && \boldsymbol{\delta}^{i,j} = \bigcirc_{k \in M_i} [\delta_{i,j,k,l} \ : \ l \in M_j], \ i \in J, \ j \in \mathfrak{S}_i \end{aligned}$$

`gprecedencemm` guarantees feasibility w.r.t. the following formula:

$$\begin{aligned} &\text{If } x_{i,k} = 1 \text{ and } x_{j,l} = 1 \Rightarrow s_i + \delta_{i,j,k,l} \leq s_j, \\ &\forall j \in \mathfrak{S}_i, \forall i \in J, \forall k \in M_i, \forall l \in M_j \end{aligned}$$

Furthermore, it extends the propagation and explanation generation algorithms of `sprecedencemm` for the application to GPRs in the context of multi-mode activities (see Sect. 6.4).

In the third strengthening variant, we add (6.18) to the formulation (5.14) - (5.20). The constraint (6.18) is stronger than (6.17) and (6.12). Firstly, this is due to the fact, that (6.17) and (6.12) can lead to infeasible solutions w.r.t. GPRs, whereas (6.18) guarantees feasibility w.r.t. GPRs. Moreover, as outlined in Sect. 6.4, the propagation algorithms implemented in `gprecedencemm` are also stronger than the ones of `sprecedencemm` and the linear constraint handler of SCIP when applied to the special case of GPRs in the context of multi-mode activities.

Example 6.2.2. *Again, we use the data introduced in example of Sect. 2.5 to illustrate the input needed for the constraint (6.18).*

In the strengthening variant with `gprecedencemm`, the vector $\boldsymbol{\delta}$ has to be determined (see (6.19)).

Therefore, we firstly construct the vector $\boldsymbol{\delta}^{1,4}$ for the precedence relation $(1,4)$:

$$\boldsymbol{\delta}^{1,2} = [2, -5] \circ [-6, 8] = [2, -5, -6, 8]$$

In total, we have to generate the vector $\boldsymbol{\delta}^{i,j}$ for every precedence relation (i,j) and then concatenate these vectors to obtain the vector $\boldsymbol{\delta}$.

6.3 Key Principles of sprecedencemm

Before sprecedencemm applies its domain propagation and explanation generation algorithms, it determines an MPI (similar to [39] and cumulativemm in Sect. 5.2) depending on the domains of the variables $x_{i,k}$ in the current node of the BaB tree.

Again, we only integrate activity-mode combinations (i,k) with $ub(x_{i,k}) > 0$ in the evaluation of the MPI. If $ub(x_{i,k}) = 0$, i.e. $x_{i,k} = 0$, mode k cannot be chosen anymore in the succeeding nodes of the BaB tree. Thus, the respective data for mode k can be excluded in the succeeding nodes.

For sprecedencemm we only need a representative duration $d_i^{\min}$ for every activity i given as follows:

$$d_i^{\min} = \min_{k \in M_i} \{d_{i,k} : ub(x_{i,k}) > 0\} \tag{6.20}$$

which is used for the domain propagation and explanation generation algorithms in Sect. 6.3.1 and 6.3.2.

As already pointed out in Sect. 5.2, the calculation of an MPI is an important step to solve the MRCPSP as the multi-mode data is transformed to a single-mode equivalent. Thus, one can apply well-known techniques for the SRCPSP to solve the MRCPSP. The idea to calculate an MPI is a standard technique to tackle MRCPSP instances and is also, e.g., proposed by Brucker and Knust [16].

6.3.1 Domain Propagation

After the calculation of $d_i^{\min}$ from (6.20) for every activity $i \in J$ in the current node of the BaB tree, we iterate through all activity pairs (i,j), where $j \in \mathfrak{S}_i$ and try to propagate the domains of i and j or detect inconsistencies.

Therefore, we compute the following values in the current node:

$$lb_j^* = lb^l(s_i) + d_i^{\min} \tag{6.21}$$

$$ub_i^* = ub^l(s_j) - d_i^{\min} \tag{6.22}$$

$lb^l(s_i)$ and $ub^l(s_i)$ correspond to the local lower and upper bound of s_i.

An inconsistency is present, if:

$$lb_j^* > ub^l(s_j), \tag{6.23}$$

i.e. it is impossible to fulfill the precedence relations (i,j) in the succeeding nodes of the BaB tree and we can cutoff the current node.

Otherwise, we try to update the domains of the variable s_i and s_j as follows:

$$\text{If } lb_j^* > lb^l(s_j): \quad lb^l(s_j) \leftarrow lb_j^* \tag{6.24}$$

$$\text{If } ub_i^* < ub^l(s_i): \quad ub^l(s_i) \leftarrow ub_i^* \tag{6.25}$$

6.3.2 Explanation Generation

To benefit from the SCIP-intern CA mechanism as in the case of the constraint handler `cumulativemm`, we have to explain inconsistencies (6.23) and domain updates (see (6.24) and (6.25)) detected by `sprecedencemm`.

Now assume, that (i, j) is a precedence relation for which (6.23) holds at the BCI c (see also Sect. 5.2.1). For the explanation of the latter inconsistency, we determine $d_i^{\min,c}$ from (6.20), $lb_j^{*,c}$ from (6.21) and the sets U_i^c for the time point c as follows:

$$U_i^c = \{m : m \in M_i \text{ and } ub^c(x_{i,m}) > 0\}. \tag{6.26}$$

The reasons for the inconsistency are the bounds of the variables s_i and s_j and the duration $d_i^{\min,c}$ at the BCI c:

$$\left([\![s_i \geq lb_j^{*,c} - d_i^{\min,c}]\!] \wedge \bigwedge_{m \in M_i \setminus U_i^c} [\![x_{i,m} = 0]\!] \right) \wedge [\![s_j \leq lb_j^{*,c} - 1]\!] \tag{6.27}$$

In other words, if (6.27) becomes true because of the domains of the variables s_i, s_j and $x_{i,k}$, $k \in M_i$ in a node of the BaB tree, SCIP knows that this leads to an inconsistency and prunes the current node of the BaB tree.

Furthermore, the CA mechanism of SCIP also asks for explanations of bound changes which were processed through `sprecedencemm`. In concrete, we have to provide an explanation for the new lower bound $lb_j^{*,e}$ of the variable s_j at the BCI e to the SCIP-intern CA mechanism.

Therefore, we assume that the precedence relation (i, j) led to the latter lower bound update.

Then the explanation is as follows:

$$[\![s_i \geq lb_j^{*,e} - d_i^{\min,e}]\!] \wedge \bigwedge_{m \in M_i \setminus U_i^e} [\![x_{i,m} = 0]\!] \implies [\![s_j \geq lb_j^{*,e}]\!] \tag{6.28}$$

(6.28) contains the minimum lower bound of the variable s_i and its necessary duration value of at least $d_i^{\min,e}$ units which in combination leads to the lower bound update of s_j.

Furthermore, we have to provide explanations for the upper bound change of a variable s_i from $ub^e(s_i)$ to $ub_i^{*,e}$ at the BCI e to the SCIP-intern CA mechanism (see (6.25)).

Again, we assume that the precedence relation (i, j) led to the upper bound change of s_i.

This domain update can be explained as follows:

$$[\![s_j \leq ub_i^{*,e} + d_i^{\min,e}]\!] \wedge \bigwedge_{m \in M_i \setminus U_i^e} [\![x_{i,m} = 0]\!] \implies [\![s_i \leq ub_i^{*,e}]\!] \tag{6.29}$$

All explanations (6.27), (6.28) and (6.29) can be transformed to possibly stronger explanations. Assume therefore, that $lb(s_i)$ and $ub(s_i)$ are the global lower and upper bounds of the variable s_i.

Firstly, if one of the inequalities

$$\begin{aligned} &lb_j^{*,c} - 1 \geq ub(s_j), \\ &lb_j^{*,c} - d_i^{\min,c} \leq lb(s_i), \\ &lb_j^{*,e} - d_i^{\min,e} \leq lb(s_i) \text{ or} \\ &ub_i^{*,e} + d_i^{\min,e} \geq ub(s_j) \end{aligned}$$

is fulfilled, we can omit the respective literal in (6.27), (6.28) or (6.29). This is due to the fact that it is globally true in the above case.

Secondly, we can determine the sets:

$$\begin{aligned} & B_i^t = \{m : m \in M_i \text{ and } d_{i,m} \geq d_{i,\text{new}}^{\min,t}\},\ t = c, e, \\ \text{where} \quad & d_{i,\text{new}}^{\min,c} = \min\{ub(s_j); lb_j^{*,c} - 1\} - \max\{lb_j^{*,c} - d_i^{\min,c}; lb(s_i)\} + 1, \\ & d_{i,\text{new}}^{\min,e} = lb_j^{*,e} - \max\{lb_j^{*,e} - d_i^{\min,e}; lb(s_i)\}, \\ \text{or} \quad & d_{i,\text{new}}^{\min,e} = \min\{ub(s_j); ub_i^{*,e} + d_i^{\min,e}\} - ub_i^{*,e}. \end{aligned}$$

We can replace

$$\bigwedge_{m \in M_i \setminus U_i^t} [\![x_{i,m} = 0]\!], t = c, e \tag{6.30}$$

by

$$\bigwedge_{m \in M_i \setminus B_i^t} [\![x_{i,m} = 0]\!], t = c, e. \tag{6.31}$$

Because of the evaluation of $d_i^{\min,t}$ through (6.20) and as $d_i^{\min,t} \geq d_{i,\text{new}}^{\min,t}$, it holds that:

$$U_i^t \subseteq B_i^t,\ \forall t = c, e.$$

Thus, in the case that $U_i^t \subset B_i^t$ for $t = c$ or $t = e$, (6.31) contains less literals than (6.30) and is therefore stronger.

6.4 Key Principles of `gprecedencemm`

The propagation rules for the starting time variables s_i and s_j belonging to the precedence relation (i, j) are similar to the ones outlined in Sect. 6.3.1 for `sprecedencemm`, with the only difference that $d_i^{\min}$ has to be substituted by $\delta_{i,j}^{\min}$ given as follows:

$$\delta_{i,j}^{\min} \quad = \quad \min_{k \in M_i, l \in M_j} \{\delta_{i,j,k,l} : ub(x_{i,k}) > 0 \wedge ub(x_{j,l}) > 0\}.$$

In concrete, we consider all possible minimal time lags belonging to the precedence relation (i,j) in the current node of the BaB tree. If e.g. it holds that $ub(x_{i,k}) = 0$ or $ub(x_{j,l}) = 0$ for a special mode combination (k,l), $k \in M_i$, $l \in M_j$, the respective minimal time lag $\delta_{i,j,k,l}$ cannot occur for (i,j) in the nodes succeeding the recent node of the BaB tree and is thus not considered in (6.4).

Assume now, that all variables s_i and $x_{i,k}$ belonging to the strengthening scenario with `sprecedencemm` have the same domains as their counterpart in the strengthening scenario with `gprecedencemm` in a node of the BaB tree. Then, it holds, that $\delta_{i,j}^{\min} \geq d_i^{\min,*}$. Thus, the propagation algorithms applied for the model with `gprecedencemm` potentially lead to stronger domain reductions than the ones applied for the model with `sprecedencemm` for an equivalent input.

The explanations generated by `gprecedencemm` in case of a conflict, a lower or an upper bound update are also rather similar to the ones generated by `sprecedencemm` (see Sect. 6.3.2) except from the fact that we again substitute $d_i^{\min,t}$ by $\delta_{i,j}^{\min,t}$ and furthermore $U_i^t, t = c,e$ by $U_{i,j}^t, t = c,e$ where:

$$U_{i,j}^t = \{(k,l) : k \in M_i, l \in M_j \text{ and } ub^t(x_{i,k}) > 0 \wedge ub^t(x_{j,l}) > 0\}, \quad t = c,e.$$

In total, the explanations of `gprecedencemm` for a conflict, a lower and an upper bound update are given by (6.32), (6.33) and (6.34), respectively:

$$\begin{aligned}
&[\![s_i \geq lb_j^{*,c} - \delta_{i,j}^{\min,c}]\!] \wedge \bigwedge_{(k,l)\in(M_i\times M_j)\setminus U_{i,j}^c} ([\![x_{i,k} = 0]\!] \vee [\![x_{j,l} = 0]\!]) \\
&\wedge [\![s_j \leq lb_j^{*,c} - 1]\!] \Longrightarrow \texttt{false} \qquad (6.32) \\
&[\![s_i \geq lb_j^{*,e} - \delta_{i,j}^{\min,e}]\!] \wedge \bigwedge_{(k,l)\in(M_i\times M_j)\setminus U_{i,j}^e} ([\![x_{i,k} = 0]\!] \vee [\![x_{j,l} = 0]\!]) \\
&\Longrightarrow [\![s_j \geq lb_j^{*,e}]\!] \qquad (6.33) \\
&[\![s_j \leq ub_i^{*,e} + \delta_{i,j}^{\min,e}]\!] \wedge \bigwedge_{(k,l)\in(M_i\times M_j)\setminus U_{i,j}^e} ([\![x_{i,k} = 0]\!] \vee [\![x_{j,l} = 0]\!]) \\
&\Longrightarrow [\![s_i \leq ub_i^{*,e}]\!] \qquad (6.34)
\end{aligned}$$

Note, that the concepts introduced in Sect. 6.3.2 to strengthen the explanations of `sprecedencemm` can be straightforwardly generalized for the use in `gprecedencemm`.

6.5 Computational Experiments

As in Sect. 4.3 and 5.3, we carried out our evaluations on the Vienna Scientific Cluster (VSC). Details about the system architecture can be found in Sect. 4.3.

For the implementation of the constraint handlers `sprecedencemm` and `gprecedencemm`, and the formulation and solution of the different models we used SCIP 3.1.1 in combination with the programming languages `C/C++`[1] .

We compare four different models denoted by the following abbreviations:

`SCIPNoStr` denotes the model (5.14) - (5.20), i.e. the model with no strengthening.

`SCIPLinStr` denotes the model (5.14) - (5.20) with the linear constraint (6.12).

`SCIPWeStr` denotes the model (5.14) - (5.20) with (6.17)[2].

`SCIPExStr` denotes the model (5.14) - (5.20) with (6.18)[3].

Again, as in Sect. 5.3, for every model we distinguish between two solution approaches, a single-core (`Max`) and a multi-core (`Best`) procedure.

In total, we apply eight different solution approaches to 810 benchmark instances with 30, 50 and 100 activities generated by Schwindt [78][4]. Hereby, the solution process of SCIP was aborted after the time limit of 1h, 2h and 4h, respectively.

Moreover, we compare the two best solution approaches belonging to the best model to the state-of-the-art exact approach for the MRCPSP with GPRs of Heilmann [39] (`BABH`).

The best known makespans, i.e. upper bounds reported online [63] were computed by `BABH` and the heuristic approach of Nonobe and Ibaraki [61]. The best known lower bounds were evaluated by Brucker and Knust [15] and `BABH`.

We compare our six approaches based on the following measurements:

`#feas` Number of instances where a feasible solution could be found within the given time limit.

`#opt` Number of instances solved to optimality within the given time limit.

`#best` Number of solutions whose makespan equals or improves the best known makespan reported online [63].

t_{tot} Average solution time in seconds for all considered instances (we take the minimal solution time of all processors for one instance in the parallel approach).

[1] Note, that for both `sprecedencemm` and `gprecedencemm` we implemented the explanation generation variant which uses the potentially stronger explanations (see Sect. 6.3.2). Preliminary experiments have shown that this version is slightly better.

[2] Here, we add `sprecedencemm` to strengthen our model. Note, that both (6.12) and (6.17) cannot be used without (6.4) to model GPRs as in this case it is possible to obtain infeasible solutions w.r.t. GPRs. Therefore, in both scenarios we speak of a "weak" formulation of GPRs.

[3] GPRs can be enforced by (6.18) in a valid way, i.e. it can be used in a standalone manner to model GPRs. Therefore, we speak of an "exact" formulation of GPRs through `gprecedencemm`.

[4] These are available online [63].

$G_{opt}(\%)$ The average optimality gap evaluated by SCIP for the instances where a feasible solution was found (we take the optimality gap of the processor which could prove optimality or otherwise could find the best makespan in the parallel approach).

`#Imp` Number of instances where we could improve the best known makespan reported online [63].

Table 6.1 shows the results for the 30-activity instances.

With all approaches except `SCIPNoStrMax`, we can solve all 270 30-activity instances to proven optimality with average solution times smaller than 59s. Moreover, we can improve the best known makespan for nine instances.

We can see that all strengthening variants lead to a more efficient mathematical model. In all single-core (multi-core) approaches, `SCIPLinStrMax(Best)` `SCIPWeStrMax(Best)` and `SCIPExStrMax(Best)`, we observe significantly smaller average solution times than with `SCIPNoPrecMax(Best)`. Furthermore, with all single-core approaches except `SCIPLinStrMax`, we can solve all instances to proven optimality.

The best results are obtained with the model `SCIPExStr`. `SCIPExStrMax(Best)` is on average at least 1.5 times as fast as `SCIPWeStrMax(Best)` and `SCIPLinStrMax(Best)`. This observation is rather intuitive, as (6.18) is stronger than (6.17) and (6.12) w.r.t. feasibility checking and propagation.

No clear conclusion can be stated about the performance of `SCIPWeStr` compared to `SCIPLinStr` for this instance set. `SCIPWeStrBest` is on average approximately 5s faster than `SCIPLinStrBest` whereas `SCIPLinStrMax` is on average approximately 6s faster than `SCIPWeStrMax`.

	#feas	#opt	#best	t_{tot}	$G_{opt}(\%)$	#Imp
`SCIPNoStrMax`	270	267	268	169.77	1.44	8
`SCIPNoStrBest`	270	270	270	58.61	0.0	9
`SCIPLinStrMax`	270	270	270	49.44	0.0	9
`SCIPLinStrBest`	270	270	270	20.71	0.0	9
`SCIPWeStrMax`	270	270	270	55.66	0.0	9
`SCIPWeStrBest`	270	270	270	15.88	0.0	9
`SCIPExStrMax`	270	270	270	32.31	0.0	9
`SCIPExStrBest`	270	270	270	9.94	0.0	9

Table 6.1: 30 Jobs: Results of our approaches

Table 6.2 compares our approaches obtained with the best model `SCIPExStr` to the state-of-the-art exact approach `BABH` w.r.t. `#opt` for different time limits.

`BABH` was tested on an Intel Pentium III 800 MHz personal computer with 256 MB SDRAM running under the operating system Windows 2000. To guarantee a fair comparison w.r.t. the processors used, we decrease the time limits for our approaches by a factor of approximately 37.7. This factor is based on the values of both processors documented in an online CPU benchmark [20].

We can conclude that for this instance set BABH outperforms our approaches. However, our parallel approach SCIPExStrBest is highly competitive to BABH for the highest time limit.

	0.027s	0.27s	2.7s	27s
SCIPExStrMax	0	0	63	238
SCIPExStrBest	0	44	208	256
	1s	10s	100s	1000s
BABH	158	198	238	257

Table 6.2: 30 Jobs: Comparison to BABH [39] w.r.t. #opt

Table 6.3 shows the results for the 50-activity instances. Again, all strengthening variants lead to a more efficient mathematical formulation. Here, the benefit is even more significant. For example, with the parallel approach SCIPExStrBest we can solve 9 more instances to proven optimality and are on average at least three times as fast as SCIPNoPrecBest.

Moreover, again regarding solution time and quality it is best to implement the model SCIPExStr which integrates the exact formulation of GPRs through gprecedencemm apart from the fact that SCIPExStrMax can solve 2 and 1 less instances to optimality than SCIPWeStrMax and SCIPLinStrMax, respectively.

Furthermore, for this instance set the model SCIPWeStr should be preferred over SCIPLinStr.

	#feas	#opt	#best	t_{tot}	$G_{opt}(\%)$	#Imp
SCIPNoStrMax	270	243	256	1570.5	16.03	51
SCIPNoStrBest	270	257	269	705.94	4.12	58
SCIPLinStrMax	270	256	265	658.84	2.98	57
SCIPLinStrBest	270	263	270	331.3	1.56	61
SCIPWeStrMax	270	257	265	564.15	2.62	58
SCIPWeStrBest	270	265	269	315.4	0.89	62
SCIPExStrMax	270	255	268	540.51	1.03	58
SCIPExStrBest	270	266	270	230.87	0.59	63

Table 6.3: 50 Jobs: Results of our approaches

In Table 6.4, we compare our best approaches SCIPExStrMax and SCIPExStrBest to BABH on the 50-activity instances. BABH outperforms SCIPExStrMax and SCIPExStrBest for the first three time limits. However, the single-core approach SCIPExStrMax and the parallel approach SCIPExStrBest produce better results for the highest time limit. With SCIPWeStrMax and SCIPWeStrBest, we can solve three and 37 more instances to optimality than BABH in this scenario, respectively.

The poorer results of our approaches for smaller time limits can be explained by the initialization phase needed until the SAT Solving techniques of SCIP become efficient. To deduce strong nogoods and large backjumps, a CA mechanism needs a

certain amount of information about the problem at hand. That is the more problem-specific explanations are given to the CA mechanism, the better are its deductions. Similarly, it takes time until the weights used in the VSIDS-based branching rule become meaningful. Thus, in the beginning the SCIP solution procedure is in a learning phase in which it behaves rather poor. After a certain period of time, SCIP has enough problem-specific information and behaves significantly better, as can be seen in the Tables 6.2 and 6.4. `BABH` uses problem-specific nogoods, also known as dominance rules, and problem-specific branching techniques from the start. Hence, the latter approach has an advantage for the smaller time limits.

	0.027s	0.27s	2.7s	27s
`SCIPExStrMax`	0	0	0	187
`SCIPExStrBest`	0	0	83	221
	1s	10s	100s	1000s
`BABH`	43	125	160	184

Table 6.4: 50 Jobs: Comparison to `BABH` [39] w.r.t. `#opt`

Finally, Table 6.5 shows our results for the 100-activity instances, the instance set of Schwindt [78] integrating the largest number of activities. Note, that to our knowledge the exact approach of Heilmann [39] was not tested on these instances.

Again, we can significantly improve the original mathematical model by our strengthening concepts. As a highlight, we point out that the approach `SCIPExStrMax` solves 160 more instances to optimality than `SCIPNoStrMax` and is on average approximately three times as fast.

Moreover, with the parallel approach `SCIPExStrBest`, we can solve approximately 73% of the instances to optimality with an average solution time of 4006s. Furthermore, we can improve the best known makespans reported online [63] for 199 instances. Only four out of 270 best known solutions had been proven to be optimal before. With `SCIPExStrBest`, we could find the optimal solution and prove its optimality for 194 more instances within the predefined time limit of 4h.

In this case, `SCIPExStr` is clearly the best model. Furthermore, the model `SCIPWeStr` integrating `sprecedencemm` is significantly better than the model integrating the linear constraints (6.12). The only bottleneck of the approaches integrating `sprecedencemm` and `gprecedencemm` for this instance set is that they cannot determine a feasible solution for all instances. Perhaps, the implementation of a problem-specific primal heuristic for SCIP can overcome this disadvantage.

Detailed benchmark files for the 30-, 50- and 100-activity instances can be found online [63]. These also integrate our results on the instances, where we could improve the best known lower or upper bounds. Moreover, the Online Resources 1, 2 and 3 contain the results obtained through `SCIPExStrBest` for the instances where we could improve the best known lower bound or makespan documented in the old benchmark files [63] for the 30-, 50- and 100-activity instances, respectively.

	#feas	#opt	#best	t_{tot}	$G_{opt}(\%)$	#Imp
SCIPNoStrMax	270	31	41	13696.5	506.78	22
SCIPNoStrBest	270	103	123	10239.0	394.44	86
SCIPLinStrMax	270	115	126	9903.61	166.11	89
SCIPLinStrBest	270	181	217	5985.62	83.28	175
SCIPWeStrMax	245	157	177	8057.41	51.59	134
SCIPWeStrBest	254	193	226	5220.3	29.27	185
SCIPExStrMax	256	191	224	4486.36	13.96	181
SCIPExStrBest	262	198	241	4005.67	11.27	199

Table 6.5: 100 Jobs: Results of our approaches

6.6 Conclusion

In this chapter, we proposed a generalization of the state-of-the-art exact algorithms for variants of the SRCPSP to the MRCPSP with GPRs. Our approach is an extension of an exact solution procedure for the MRCPSP with SPRs which was introduced in Chapt. 5. The respective SCIP-models could be straightforwardly extended for the formulation of the MRCPSP with GPRs.

Moreover, we introduced two new global constraints `sprecedencemm` and `gprecedencemm` with which one can model SPRs and GPRs in the context of multi-mode activities, respectively. The latter integrate constraint propagation and explanation generation specially tailored to the above problem characteristics.

We analyzed four different mathematical models for the MRCPSP with GPRs. The first is the straightforward extension of the best SCIP-model presented in Chapt. 5. The other three models integrate different constraints to potentially strengthen the original formulation. One integrates linear constraints to obtain a stronger LP-relaxation. The other two additionally contain `sprecedencemm` and `gprecedencemm` to model GPRs such that problem-specific knowledge is integrated into the solution process. In two models (`SCIPLinStr` and `SCIPWeStr`) the additional constraint is weaker than in the last model (`SCIPExStr`) with `gprecedencemm`. More precisely, solutions which are feasible w.r.t. the constraint handler `gprecedencemm` employed in `SCIPExStr`, also fulfill the additional constraints added in `SCIPLinStr` and `SCIPWeStr`, but not vice versa.

Our computational results show, that the proposed strengthening variants significantly improve the original mathematical formulation on all benchmark sets. The best model is the one integrating `gprecedencemm`.

Moreover, one can observe that especially for the 50- and 100-activity instances it is clearly better to strengthen the original formulation by `sprecedencemm` and `gprecedencemm` than by the linear constraints (6.12). This can be explained by the fact, that our constraint handlers integrate problem-specific propagation and explanation generation algorithms. The linear constraint handler of SCIP integrates more general techniques which potentially lead to smaller search space reductions.

We also compared a single-core and a parallel solution approach including the best model `SCIPExStr` to the state-of-the-art exact algorithm of Heilmann [39] (`BABH`). For the three smallest imposed time limits, `BABH` can solve significantly more instances to optimality. However, if the time limits and instances become larger, we outperform the latter approach. E.g. for instances with 50 activities we can solve 37 more instances to optimality within a comparable time limit. This is explained by a possibly time-consuming learning phase of the SCIP solution procedure until the CP-SAT techniques become efficient. We assume, that when it comes to closing hard instances and when larger time limits are allowed, our approach should be preferred over `BABH`.

Furthermore, our solution approach is more flexible as the applied models can be extended or changed and we still can use the solution algorithm provided by SCIP. In contrast, the extension of the specialized approach of Heilmann [39] might not be straightforward when a new consideration has to be taken into account.

Our parallel approach is dependent on the makespan of a feasible solution. However, if there is no feasible solution at hand, our single-core approach can be applied as it uses a pre-computable upper bound for the makespan as input. Our computational experiments show that also this relatively high upper bound as input leads to competitive results.

Furthermore, the results for the 100-activity instances, the test set from the literature containing the largest problem instances, are highly encouraging. We could find the optimal solution and prove its optimality for 198 problem instances. To our knowledge, only the best solutions of four instances were known to be optimal before. In addition, we could improve the best known makespan for 199 instances.

In summary, we could close 289 open instances and could improve the best known makespan for 271 instances from the benchmark of Schwindt [78] with our best performing solution approach.

Chapter 7

A heuristic for the MRCMPSP with SPRs

For the MISTA competition [90], we implemented a model within SCIP which applies our constraint handler `cumulativemm` (see Sect. 5.2) for the modeling and solution of the MRCMPSP with SPRs with the objective to lexicographically minimize first the total project delay (TPD) and then the total makespan (TMS). We are able to solve the competition instances with at most two projects to optimality by the solution of the model from Sect. 7.1 in less than 15s. For the remaining instances we implemented a local search algorithm (see Sect. 7.2) whereas the construction and improvement step are based on the solution of variations of the model of Sect. 7.1. The results obtained by the exact and heuristic algorithm can be found in Sect. 7.3.

7.1 A SCIP-model for the MRCMPSP with SPRs

When using our `cumulativemm`-constraint and the notation provided by the MISTA competition organizers [90], the MRCMPSP to be considered can be formulated as

follows within SCIP:

$$\min \quad H_{\max} \cdot \left(\sum_{i \in P} s_{i,|J_i|+1,1} - \sum_{i \in P} (r_i + CPD_i) \right) + s_g \tag{7.1}$$

$$s.t. \quad \sum_{k \in M_{i,j}} x_{i,j,k} = 1,$$

$$\forall i \in \mathcal{P},\ \forall j \in J_i \cup \{0, |J_i|+1\} \tag{7.2}$$

$$s_{i,j,k} + d_{i,j,k} \cdot x_{i,j,k} \leq s_{i,j',k'},$$

$$\forall i \in \mathcal{P},\ j \prec j',\ k \in M_{i,j},\ k' \in M_{i,j'} \tag{7.3}$$

$$s_{i,|J_i|+1,1} \leq s_g, \qquad \forall i \in \mathcal{P} \tag{7.4}$$

$$\sum_{j \in J_i} \sum_{k \in M_{i,j}} r^{\nu}_{i,j,k,l} \cdot x_{i,j,k} \leq c_{i,l}, \qquad \forall i \in \mathcal{P},\ l \in L^{\nu}_i \tag{7.5}$$

$$\texttt{cumulativemm}(\hat{\mathbf{s}}^i, \hat{\mathbf{x}}^i, \hat{\mathbf{d}}^i, \hat{\mathbf{r}}^i_l, c_{i,l}), \qquad \forall i \in \mathcal{P}, l \in L^{\rho}_i \tag{7.6}$$

$$\texttt{cumulativemm}(\hat{\mathbf{s}}, \hat{\mathbf{x}}, \hat{\mathbf{d}}, \hat{\mathbf{r}}_g, c_g), \qquad \forall g \in G^{\rho} \tag{7.7}$$

$$s_{i,j,k} \in \{lb(s_{i,j,k}), \ldots, ub(s_{i,j,k})\},$$

$$x_{i,j,k} \in \{0,1\},$$

$$\forall i \in \mathcal{P},\ j \in J_i \cup \{0, |J_i|+1\},\ k \in M_{i,j} \tag{7.8}$$

where

$$\hat{\mathbf{s}}^i = \mathbf{s}^{i,0} \circ \cdots \circ \mathbf{s}^{i,|J_i|+1}, \qquad \text{where } \mathbf{s}^{i,j}_k = s_{i,j,k},$$

$$\forall\, i \in \mathcal{P},\ j \in J_i \cup \{0, |J_i|+1\}, k \in M_{i,j} \tag{7.9}$$

$$\hat{\mathbf{x}}^i = \mathbf{x}^{i,0} \circ \cdots \circ \mathbf{x}^{i,|J_i|+1}, \qquad \text{where } \mathbf{x}^{i,j}_k = x_{i,j,k},$$

$$\forall\, i \in \mathcal{P},\ j \in J_i \cup \{0, |J_i|+1\}, k \in M_{i,j} \tag{7.10}$$

$$\hat{\mathbf{d}}^i = \mathbf{d}^{i,0} \circ \cdots \circ \mathbf{d}^{i,|J_i|+1}, \qquad \text{where } \mathbf{d}^{i,j}_k = d_{i,j,k},$$

$$\forall\, i \in \mathcal{P},\ j \in J_i \cup \{0, |J_i|+1\}, k \in M_{i,j} \tag{7.11}$$

$$\hat{\mathbf{r}}^i_l = \mathbf{r}^{i,0,l} \circ \cdots \circ \mathbf{r}^{i,|J_i|+1,l}, \qquad \text{where } \mathbf{r}^{i,j,l}_k = r^{\rho}_{i,j,k,l},$$

$$\forall\, i \in \mathcal{P},\ j \in J_i \cup \{0, |J_i|+1\}, k \in M_{i,j}, l \in L^{\rho}_i \cup G^{\rho} \tag{7.12}$$

and

$$\hat{\mathbf{s}} = \hat{\mathbf{s}}^0 \circ \cdots \circ \hat{\mathbf{s}}^{n-1} \tag{7.13}$$

$$\hat{\mathbf{x}} = \hat{\mathbf{x}}^0 \circ \cdots \circ \hat{\mathbf{x}}^{n-1} \tag{7.14}$$

$$\hat{\mathbf{d}} = \hat{\mathbf{d}}^0 \circ \cdots \circ \hat{\mathbf{d}}^{n-1} \tag{7.15}$$

$$\hat{\mathbf{r}}_l = \hat{\mathbf{r}}^0_l \circ \cdots \circ \hat{\mathbf{r}}^{n-1}_l \tag{7.16}$$

$$\hat{\mathbf{r}}_g = \hat{\mathbf{r}}^0_g \circ \cdots \circ \hat{\mathbf{r}}^{n-1}_g \tag{7.17}$$

Note that the variables $s_{i,j,k}$ represent the starting times of the activity $j \in J$ of project $i \in \mathcal{P}$ in mode $k \in M_{i,j}$ and the variables $x_{i,j,k}$ model the processing of the

activity $j \in J$ of project $i \in \mathcal{P}$ in mode $k \in M_{i,j}$. Hereby, $s_{i,|J_i|+1,1}$ are the starting time variables for the dummy jobs representing the ending of project $i \in \mathcal{P}$ with $M_{i,|J_i|+1} = \{1\}$.

The objective function (7.1) consists of a combination of the TPD and the TMS s_g, which is evaluated through (7.4). Thereby, we multiply the TPD with a sufficiently big makespan H_{max} of the complete multi-project instance to guarantee the lexicographical optimization of the two values.

(7.3) are multi-mode precedence constraints. With (7.5), we assure that the non-renewable resource limits are not exceeded.

In (7.6) and (7.7), we apply our `cumulativemm`-constraint to model the resource constraints for the local and global renewable resources. Thereby, we use the variable vectors $\hat{\mathbf{s}}^i$, $\hat{\mathbf{s}}$, $\hat{\mathbf{x}}^i$ and $\hat{\mathbf{x}}$. In this context, the operator $\circ$ is defined as in the Sections 5.1 and 6.2.

7.2 Local Search for the MRCMPSP with SPRs

Our local search algorithm consists of a construction and an improvement step. In the construction step, we firstly try to solve every single project separately with SCIP with the aim of makespan minimization. After a certain time limit L, we stop the solution process and store the currently best MRCPSP solutions. Then, we try to find the best sequence according to which the projects are ordered. For this we solve a one-machine problem with given release dates and possible preemption with the aim of minimizing the complete project delay (equivalent to the first term in (7.1)). From the optimal sequence and the starting times of the solutions of every single MRCPSP project, we generate a feasible schedule for the MRCMPSP. Based on this schedule, we generate an activity list, which sorts all MRCMPSP activities in ascending order of their starting times. After applying the SSGS (see [16, p.144 ff.] and Sect. 2.5) to the latter activity list, we obtain a feasible starting solution for our local search algorithm.

Our improvement step consists of the search for a new solution in one of the four following neighborhoods. Two neighborhoods are based on the relaxation of at most 2 projects in the currently best schedule. For the first one we randomly relax 1-2 projects and for the second one we relax 1-2 projects for which the relaxation would lead to the smallest lower bound on the objective function (7.1).
The third neighborhood is based on the relaxation of a random number of activities which are scheduled at the end in the incumbent solution.

For the fourth neighborhood we only relax the mode variables in the incumbent solution. With this neighborhood we try to find a mode-assignment which minimizes the complete renewable resource energy [8, p.15 ff.]. After having found a new mode-assignment, we apply the SSGS to the incumbent solution with the new mode-assignment to obtain a new feasible schedule.
The search for a new schedule in the neighborhoods is based on solving a variation of the model presented in Sect. 7.1 with SCIP within a given time limit. Note, that

we store information about the models already solved, so that we do not solve the same model twice.

If a solution in one of the neighborhoods has a better objective value than the incumbent solution, we update the best solution. Note that the neighborhoods are randomly chosen, whereas we assign a higher selection probability to the first two neighborhoods.

7.3 Results

Our exact and heuristic approach was tested on the Vienna Scientific Cluster (VSC) (see Sect. 4.3). We implemented our approaches in `C/C++` in combination with SCIP 3.0.0.

The Tables 7.1 and 7.2 show the results for the instance sets `A` and `B` provided online by the competition organizers [19]. One can see the average TPD and TMS, the best TPD and TMS (belonging to the best TPD) based on 10 runs for every instance. Moreover, the tables contain the total solution times (t_{tot}) and the gaps to the best known solutions (G_{BK}) for the instance set `A`. Note, that for the instances `A-1-A-3`, we found the optimal solution in less than 15s by solving the model of Sect. 7.1 with SCIP. For the remaining instances, we applied the heuristic algorithm of Sect. 7.2 with the time limit $t_{tot} = 300s$ given by the competition organizers. Overall, our solution procedure was ranked ninth among all competition participants.

More detailed comparisons including all solution approaches are provided by Wauters et al. [90].

	avg. TPD	avg. TMS	best TPD	best TMS	t_{tot} (s)	G_{BK} (%)
`A-1`	1.00	23.00	1	23	0.67	0.00
`A-2`	2.00	41.00	2	41	14.35	0.00
`A-3`	0.00	50.00	0	50	4.83	0.00
`A-4`	74.73	54.82	65	45	300	0.00
`A-5`	208.11	123.44	192	121	300	25.49
`A-6`	273.90	132.00	221	114	300	50.34
`A-7`	689.60	208.30	676	204	300	13.42
`A-8`	433.60	172.80	392	168	300	29.80
`A-9`	404.20	159.40	370	158	300	65.92
`A-10`	1313.50	369.80	1286	368	300	32.71
						21.77

Table 7.1: Results for the instance set `A`

7.4 Conclusion

For the MISTA competition 2013 [90], we applied the constraint `cumulativemm` to exactly and heuristically solve the MRCMPSP with SPRs. We are able to solve the

	avg. TPD	avg. TMS	best TPD	best TMS	t_{tot} (s)
B-1	455.60	145.50	441	142	300
B-2	549.10	173.60	520	170	300
B-3	743.80	229.70	727	222	300
B-4	1832.10	328.40	1689	309	300
B-5	1321.90	303.30	1203	289	300
B-6	1290.80	265.80	1222	252	300
B-7	1418.40	297.10	1321	291	300
B-8	3848.90	599.80	3713	604	300
B-9	7600.20	1059.30	6993	1022	300
B-10	3634.70	468.10	3422	466	300

Table 7.2: Results for the instance set B

competition instances with at most two projects to optimality. For the remaining instances, we implemented a local search algorithm which relaxes parts of the complete problem and tries to solve the remaining problem to optimality with SCIP. With an increasing number of projects, even the exact solution of the relaxed problems can take a large amount of time. Thus, we assume a potential for an ample improvement of the results by relaxing greater parts of the incumbent solutions and optimizing the relaxed problems by metaheuristics like e.g. genetic algorithms.

This chapter and especially Sect. 7.1 shows, that with SCIP and our constraint handlers at hand, it is in many cases straightforward to model more complex variants of the RCPSP, i.e. to generalize the CP-SAT approaches proposed in the preceding chapters to other problem classes which can be found in the area of project scheduling.

Chapter 8

Conclusion and future research

State-of-the-art exact approaches for variants of the SRCPSP consist of a BaB algorithm combining CP and SAT Solving techniques (see [76], [77] and [75]).

As a first contribution, in Chapt. 4, we provided a detailed computational analysis of the performance variability which can be detected in recent CP-SAT approaches for the SRCPSP with SPRs. We found out that varying the initial domains based on different predefined upper bounds leads to a significant variation in the behavior of state-of-the-art CP-SAT approaches. Moreover, a run with a higher upper bound can result in better results than a run with a smaller upper bound.

This effect was explained by the differing conflict statistics used for the SAT Solving based branching heuristics for the different initial domains. When a more conservative branching heuristic was applied, the above effect could not be observed. Varying conflict statistics can direct the BaB algorithm to completely different search paths. Thus, also a run with a high upper bound can lead to excellent results. That is, also in the latter case, due to beneficial branching decisions a good feasible solution can be found early, or strong nogoods or large backjumps can be deduced.

As it is hard to predict the search path for a certain upper bound beforehand, we proposed a parallel approach to exploit the observed effect. The computational experiments show, that especially for non-easy instances, the parallel approaches significantly outperform the single-core approaches.

Still, there are many interesting future research directions with regard to the evaluations in this chapter. Schutt et al. [74] propose an initialization phase for the conflict statistics in the CP-SAT approach LCG where branching is based on the SSGS. The aim is, to improve the initial activity counters for the VSIDS branching heuristic for the problem at hand. It would be interesting to also implement a problem specific initialization phase of the conflict statistics and to measure the effect of varying the initial domains in this scenario.

Furthermore, more information should be collected in the course of the solution algorithm, e.g. the number of nodes, backtracks, detected conflicts etc. should be measured, to perhaps explain the observed effect in a more detailed way.

Performance variability can also be observed in MIP solvers. Fischetti and Monaci [29] provide a detailed computational analysis in this context. They discuss many initial conditions causing performance variability and again propose a parallel approach to exploit the observed erraticism. Firstly, it would be interesting to also analyze different initial conditions, not only the initial domains, which lead to erraticism in the CP-SAT approaches for the RCPSP. For example, these could be different initial seeds for the SCIP-internal functions which include randomness or different random orders of the constraints. In addition, an evaluation of the parallel "bet-and-run" procedure of Fischetti and Monaci [29] in our context is also of great interest. In general, a more comprehensive literature study about erraticism in CP or SAT solvers should be considered for further experiments.

The Chapters 5 and 6 contain the main contribution of this dissertation, namely the extension and application of the state-of-the-art CP-SAT approaches to RCPSP instances with multi-mode jobs. In Chapt. 5, we proposed three modeling formulations for the MRCPSP with SPRs, one for the optimization framework G12 [83] and two for the optimization framework SCIP [2]. The latter frameworks both provide a solution algorithm combining CP and SAT techniques whereat G12 integrates the state-of-the-art exact algorithm for variants of the SRCPSP, namely lazy clause generation [28].

For one SCIP-model a new constraint handler `cumulativemm` was implemented. The latter captures propagation and explanation generation algorithms specially tailored to renewable resources in the context of multi-mode jobs. The `cumulative` constraint provided by G12 integrates similar techniques [73] and can be applied to model MRCPSP instances. However, we discussed cases where our implementation has an advantage over the implementation in G12.

For every model, we tested two approaches, a single-core approach (`Max`) and a parallel approach (`Best`). Thereby, similar to Chapt. 4, `Best` takes advantage of the performance variability triggered by the variation of the initial domains based on different upper bounds. Our computational experiments show that for the 20- and 30-job instances from the PSPLIB, the G12-model has the best performance. Moreover, with both our single-core and parallel solution approach, we significantly outperform the state-of-the-art exact approach for the MRCPSP with SPRs [92]. The best performing single-core approach, i.e. when the G12-model with initial domains based on a pre-computable upper bound is solved by LCG, can solve nine more 30-job instances to proven optimality than the latter approach and is one average approximately 1.8 times as fast. With the parallel approach, significant improvements can be obtained compared to the single-core approach. We solve 15 more 30-job instances to proven optimality than Zhu et al. [92] and are on average 2.1 times as fast.

The approaches based on the SCIP-model, integrating our constraint handler `cumulativemm`, are highly competitive to the state-of-the-art exact approach of Zhu et al. [92], but outperformed by the latter on the 20- and 30-job instances.

Furthermore, we were the first to exactly solve new instances of the MRCPSP with SPRs with 50 and 100 jobs from the literature [87]. For these instances, the solution approaches based on our best SCIP-model should be preferred over the ones based on the G12-model. In total, we were able to close 324 open 50- and 100-job instances and improve the best known makespan for 176 of the 50- and 100-job instances.

The SCIP-model for the MRCPSP with SPRs was straightforwardly generalized to a formulation of the MRCPSP with GPRs in Chapt. 6. Moreover, in the latter chapter, we proposed three improved formulations extending the original model of the MRCPSP with GPRs. One formulation integrates a linear constraint to strengthen the original model. The other two integrate the constraint handlers `sprecedencemm` and `gprecedencemm`, which we implemented within SCIP. With these, one can model SPRs and GPRs in the context of multi-mode jobs, respectively. Moreover, they contain propagation and explanation generation algorithms specially tailored to the latter problem characteristics.

Regarding the propagation strength, we found out that the application of `gprecedencemm` leads to potentially stronger domain reductions than `sprecedencemm` and the latter itself leads to potentially stronger domain reductions than the linear constraints when applied for the strengthening of the original formulation. Computational experiments on instances from the literature [78] supported these findings. The model integrating `gprecedencemm` performs significantly better than the other two strengthening variants. Moreover, adding `sprecedencemm` should be preferred over adding the proposed linear constraints to the original formulation.

Again, for every model, we analyzed a single-core solution approach and a parallel solution approach, which exploits erraticism which can be detected for varying initial domains. With both the single-core and the parallel approach integrating the best performing model, we outperform the state-of-the-art exact algorithm for the MRCPSP with GPRs [39] on instances with 50 jobs when imposing time limits of 27s. With the single-core and the parallel approach, we can solve three and 37 more 50-job instances to proven optimality within the latter time limit. However, when rather small time limits are given or if the 30-job instances are considered, the exact approach of Heilmann [39] should be preferred over our approaches. Nonetheless, the reader should keep in mind that our approaches are more flexible than the latter approach when it comes to extensions of the considered problem classes.

Another important contribution of Chapt. 6 is the closing of 289 open instances with 30, 50 and 100 jobs and the improvement of the best known makespan for 271 instances with our best performing approach.

Chapt. 7 was motivated by the MISTA competition 2013 [90]. It shows the flexibility of our solution concepts, when one is confronted with more general multi-mode problem classes. We provide a model for the MRCMPSP with SPRs integrating our constraint `cumulativemm`. Small instances could be exactly by the CP-SAT approaches provided by SCIP. For larger problem instances provided by Wauters

et al. [90], we applied a local search heuristic which iteratively solves variants of the original formulation.

The implementation of our constraint handlers `cumulativemm`, `sprecedencemm` and `gprecedencemm` can still be strengthened. For example, one could integrate new propagation and respective explanation generation algorithms into `cumulativemm` as at the moment it solely captures TP. The combination of TP and EF proposed by Vilím [89] has led to improvements compared to using solely TP when implemented in a CP-SAT context for the SRCPSP with SPRs [76]. Therefore, this should be the first extension to be tested.

The integration of MIP techniques into our constraint handlers is another interesting future research direction. At the moment our constraints only capture problem-specific propagation and explanation generation algorithms, i.e. the problem specific bound updates and cutoffs are only based on the latter techniques. There is a potential for further improvements by a strengthening of the node-specific LP relaxation. Therefore, one could think about functions for our constraint handlers which separate problem-specific cuts for the SCIP-internal solution procedure. A first step could be the integration of the cutting plane generation principles for the MRCPSP with SPRs proposed by Zhu et al. [92]. Moreover, different MIP representations for renewable resource constraints, SPRs and GPRs could be stored in the respective constraint handlers and added lazily when a constraint in the latter representation is violated. It is also worthwhile to test the influence of primal heuristics which frequently try to construct a feasible solution for the MRCPSP with SPRs and GPRs in nodes of the BaB tree to possibly improve the global upper bound. Furthermore, the application of our SCIP-models as the exact plugin in hybrid metaheuristics [14] is of interest when one wants to tackle instances arising in practice.

A further direction for future research is the application of our constraint handlers to model and solve other variants of the RCPSP. In this context, it would be interesting to evaluate the benefit of integrating a single-mode version of `sprecedencemm` and `gprecedencemm` for the modeling of the SRCPSP with SPRs and GPRs, respectively. Furthermore, it would be worthwhile to test the efficiency of a model for the SRCPSP with discounted cash flows (see Sect. 2.1) which integrates our constraints. Moreover, it is possible to extend `cumulativemm` so that the resource availabilities $C_k^\rho,\ k \in R$ can be variables as in the `cumulative` constraint of G12 [73]. With this extension, one could evaluate models integrating our constraint handlers for special variants of resource leveling problems (see Sect. 2.1), i.e. the resource investment problem with SPRs and GPRs.

Various COPs like e.g. timetabling problems, sports league and machine scheduling problems can be formulated as variants of the RCPSP [16]. An analysis about how the CP-SAT approaches perform in this context would be interesting. In addition to an evaluation of the performance of CP-SAT approaches for scheduling problems, they should also be developed and tested for other Operations Research problems, like e.g. for vehicle routing problems. In general, an extensive computa-

tional or theoretical study analyzing properties of COPs which positively correlate with the performance of CP-SAT approaches applied to these COPs would be of great interest. In concrete, the last topic also aims at answering the question why state-of-the-art CP-SAT approaches work that good for variants of the RCPSP but not for different problem classes.

Currently, Lamas Vilches and Demeulemeester [52] proposed a Branch and Cut algorithm based on minimal forbidden sets to solve the SRCPSP with SPRs and stochastic activity durations. One could also think of generalizations of our constraint handlers and especially `sprecedencemm` in the context of stochastic activity durations and compare the efficiency of the resulting CP-SAT approaches in combination with sample average approximation to their Branch and Cut algorithm.

To conclude, the results obtained in this dissertation regarding CP-SAT approaches for variants of the RCPSP pave the path for a number of fruitful future research directions.

Bibliography

[1] T. Achterberg. *SCIP-a framework to integrate constraint and mixed integer programming*. Konrad-Zuse-Zentrum für Informationstechnik Berlin, 2004.

[2] T. Achterberg. SCIP: solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009.

[3] A. Agarwal, S. Colak, and S. Erenguc. Metaheuristic methods. In Christoph Schwindt and Jürgen Zimmermann, editors, *Handbook on Project Management and Scheduling Vol.1*, International Handbooks on Information Systems, pages 57–74. Springer International Publishing, 2015. ISBN 978-3-319-05442-1. doi: 10.1007/978-3-319-05443-8_4. URL `http://dx.doi.org/10.1007/978-3-319-05443-8_4`.

[4] A. Aggoun and N. Beldiceanu. Extending CHIP in order to solve complex scheduling and placement problems. *Mathematical and Computer Modelling*, 17 (7):57–73, 1993.

[5] K. R. Apt. *Principles of constraint programming*. Cambridge University Press, 2003.

[6] C. Artigues, S. Demassey, and E. Néron. *Resource-constrained project scheduling: models, algorithms, extensions and applications*. John Wiley & Sons, 2013.

[7] F. Ballestín, A. Barrios, and V. Valls. Looking for the best modes helps solving the MRCPSP/max. *International Journal of Production Research*, 51(3):813–827, 2013.

[8] P. Baptiste, C. Le Pape, and W. Nuijten. *Constraint-based scheduling: applying constraint programming to scheduling problems*, volume 39. Springer Netherlands, 2001.

[9] J.-H. Bartels and J. Zimmermann. Scheduling tests in automotive R&D projects. *European Journal of Operational Research*, 193(3):805–819, 2009.

[10] M. Bartusch, R. H. Möhring, and F. J. Radermacher. Scheduling project networks with resource constraints and time windows. *Annals of Operations Research*, 16(1):199–240, 1988.

[11] T. Berthold, S. Heinz, M. E. Lübbecke, R. H. Möhring, and J. Schulz. A constraint integer programming approach for resource-constrained project scheduling. In Andrea Lodi, Michela Milano, and Paolo Toth, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 6140 of *Lecture Notes in Computer Science*, pages 313–317. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-13519-4.

[12] C. Bessière and P. Van Hentenryck. To be or not to be ... a global constraint. In Francesca Rossi, editor, *Principles and Practice of Constraint Programming – CP 2003*, volume 2833 of *Lecture Notes in Computer Science*, pages 789–794. Springer Berlin Heidelberg, 2003. ISBN 978-3-540-20202-8. doi: 10.1007/978-3-540-45193-8_54. URL `http://dx.doi.org/10.1007/978-3-540-45193-8_54`.

[13] J. Blazewicz, J. K. Lenstra, and A. H. G. Rinnooy Kan. Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*, 5(1):11–24, 1983.

[14] C. Blum, J. Puchinger, G. R. Raidl, and A. Roli. Hybrid metaheuristics in combinatorial optimization: A survey. *Applied Soft Computing*, 11(6): 4135 – 4151, 2011. ISSN 1568-4946. doi: http://dx.doi.org/10.1016/j.asoc.2011.02.032. URL `http://www.sciencedirect.com/science/article/pii/S1568494611000962`.

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

[16] P. Brucker and S. Knust. *Complex scheduling*. Springer Verlag, 2006. ISBN 3540295453.

[17] P. Brucker, A. Drexl, R. Mohring, K. Neumann, and E. Pesch. Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, 112(1):3–41, 1999.

[18] J. Coelho and M. Vanhoucke. Multi-mode resource-constrained project scheduling using RCPSP and SAT solvers. *European Journal of Operational Research*, 213(1):73 – 82, 2011. ISSN 0377-2217. doi: http://dx.doi.org/10.1016/j.ejor.2011.03.019. URL `http://www.sciencedirect.com/science/article/pii/S037722171100230X`.

[19] competition organizers. MISTA 2013 challenge: the Multi-Mode Resource-Constrained Multi-Project Scheduling Problem. `http://allserv.kahosl.be/mista2013challenge/overview.html`, 2013. [Online; accessed 04-03-2013].

[20] CPU benchmark. Passmark CPU benchmark. `https://www.cpubenchmark.net/cpu_list.php`, 2014.

[21] R. J. Dakin. A tree-search algorithm for mixed integer programming problems. *The Computer Journal*, 8(3):250–255, 1965. doi: 10.1093/comjnl/8.3.250. URL http://comjnl.oxfordjournals.org/content/8/3/250.abstract.

[22] M. Davis and H. Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, July 1960. ISSN 0004-5411. doi: 10.1145/321033.321034. URL http://doi.acm.org/10.1145/321033.321034.

[23] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, July 1962. ISSN 0001-0782. doi: 10.1145/368273.368557. URL http://doi.acm.org/10.1145/368273.368557.

[24] B. De Reyck and W. Herroelen. The multi-mode resource-constrained project scheduling problem with generalized precedence relations. *European Journal of Operational Research*, 119(2):538–556, 1999.

[25] E. L. Demeulemeester and W. S. Herroelen. New benchmark results for the resource-constrained project scheduling problem. *Management Science*, 43(11): 1485–1492, 1997. doi: 10.1287/mnsc.43.11.1485. URL http://dx.doi.org/10.1287/mnsc.43.11.1485.

[26] E. L. Demeulemeester and W. S. Herroelen. *Project scheduling: a research handbook*, volume 102. Kluwer Academic Publishers Group, 2002.

[27] S. E. Elmaghraby. *Activity networks: Project planning and control by network models*. Wiley, New York, 1977.

[28] T. Feydy and P. J. Stuckey. Lazy clause generation reengineered. In Ian P. Gent, editor, *Principles and Practice of Constraint Programming - CP 2009*, volume 5732 of *Lecture Notes in Computer Science*, pages 352–366. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-04243-0. doi: 10.1007/978-3-642-04244-7_29. URL http://dx.doi.org/10.1007/978-3-642-04244-7_29.

[29] M. Fischetti and M. Monaci. Exploiting erraticism in search. *Operations Research*, 62(1):114–122, 2014.

[30] M. Frey, C. Artigues, R. Kolisch, and P. Lopez. Scheduling and Planning the Outbound Baggage Process at International Airports. In *IEEE International Conference on Industrial Engineering and Engineering Management (IEEM 2010)*, pages pp. 101–105, Macao, China, December 2010. IEEE. doi: 10.1109/IEEM.2010.5675613. URL https://hal.archives-ouvertes.fr/hal-00561723.

[31] G12 Constraint Programming Platform. G12 Constraint Programming Platform. http://nicta.com.au/research/projects/constraint_programming_platform, 2013.

[32] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, and T. S. Woodall. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Proceedings of the 11th European PVM/MPI Users' Group Meeting*, pages 97–104, Budapest, Hungary, September 2004.

[33] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. WH Freeman & Co., San Francisco, 1979.

[34] Gecode. `http://www.gecode.org/`, 2012.

[35] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. volume 5 of *Annals of Discrete Mathematics*, pages 287 – 326. Elsevier, 1979. doi: http://dx.doi.org/10.1016/S0167-5060(08)70356-X. URL `http://www.sciencedirect.com/science/article/pii/S016750600870356X`.

[36] 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.

[37] S. Hartmann and A. Drexl. Project scheduling with multiple modes: A comparison of exact algorithms. *Networks*, 32(4):283–297, 1998.

[38] W. D. Harvey and M. L. Ginsberg. Limited discrepancy search. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 607–615, 1995.

[39] R. Heilmann. A branch-and-bound procedure for the multi-mode resource-constrained project scheduling problem with minimum and maximum time lags. *European Journal of Operational Research*, 144(2):348–365, 2003.

[40] C. Heimerl and R. Kolisch. Scheduling and staffing multiple projects with a multi-skilled workforce. *OR Spectrum*, 32(2):343–368, 2010.

[41] S. Heinz and J. Schulz. Explanations for the cumulative constraint: An experimental study. In Panos M. Pardalos and Steffen Rebennack, editors, *Experimental Algorithms*, volume 6630 of *Lecture Notes in Computer Science*, pages 400–409. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-20661-0. doi: 10.1007/978-3-642-20662-7_34. URL `http://dx.doi.org/10.1007/978-3-642-20662-7_34`.

[42] S. Heinz, W.-Y. Ku, and J. C. Beck. Recent improvements using constraint integer programming for resource allocation and scheduling. In Carla Gomes and Meinolf Sellmann, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 7874 of *Lecture Notes in Computer Science*, pages 12–27. Springer Berlin Heidelberg, 2013.

[43] W. Herroelen. Project scheduling - theory and practice. *Production and operations management*, 14(4):413–432, 2005.

[44] A. Horbach. A boolean satisfiability approach to the resource-constrained project scheduling problem. *Annals of Operations Research*, 181(1):89–107, 2010.

[45] J. Hwang and D. G. Mitchell. 2-way vs. d -way branching for CSP. In Peter van Beek, editor, *Principles and Practice of Constraint Programming - CP 2005*, volume 3709 of *Lecture Notes in Computer Science*, pages 343–357. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-29238-8. doi: 10.1007/11564751_27. URL `http://dx.doi.org/10.1007/11564751_27`.

[46] JaCoP. JaCoP - Java Constraint Programming solver. `http://jacop.osolpro.com`, 2014.

[47] R. Kolisch. Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. *European Journal of Operational Research*, 90(2):320–333, 1996.

[48] R. Kolisch and A. Drexl. Local search for nonpreemptive multi-mode resource-constrained project scheduling. *IIE Transactions*, 29(11):987–999, 1997.

[49] R. Kolisch and S. Hartmann. Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European Journal of Operational Research*, 174(1):23–37, 2006. ISSN 0377-2217.

[50] R. Kolisch and A. Sprecher. PSPLIB-a project scheduling problem library. *European Journal of Operational Research*, 96(1):205–216, 1997.

[51] O. Koné, C. Artigues, P. Lopez, and M. Mongeau. Event-based MILP models for resource-constrained project scheduling problems. *Computers & Operations Research*, 38(1):3 – 13, 2011. ISSN 0305-0548. doi: http://dx.doi.org/10.1016/j.cor.2009.12.011. URL `http://www.sciencedirect.com/science/article/pii/S0305054809003360`.

[52] P. Lamas Vilches and E. Demeulemeester. A purely proactive scheduling procedure for the resource-constrained project scheduling problem with stochastic activity durations. *Available at SSRN: http://ssrn.com/abstract=2464056 or http://dx.doi.org/10.2139/ssrn.2464056*, 2014.

[53] J. P. Marques-Silva and K. A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999.

[54] K. Marriott, N. Nethercote, R. Rafeh, P. J. Stuckey, M. G. De La Banda, and M. Wallace. The design of the Zinc modelling language. *Constraints*, 13(3): 229–267, 2008.

[55] L. Mercier and P. Van Hentenryck. Edge finding for cumulative scheduling. *INFORMS Journal on Computing*, 20(1):143–153, 2008. ISSN 1526-5528.

[56] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Annual Design Automation Conference*, DAC '01, pages 530–535, New York, NY, USA, 2001. ACM. ISBN 1-58113-297-2. doi: 10.1145/378239.379017. URL `http://doi.acm.org/10.1145/378239.379017`.

[57] mpi4py. `http://mpi4py.scipy.org/`, 2012.

[58] K. Neumann and J. Zimmermann. Procedures for resource leveling and net present value problems in project scheduling with general temporal and resource constraints. *European Journal of Operational Research*, 127(2):425–443, 2000.

[59] K. Neumann, C. Schwindt, and J. Zimmermann. *Project scheduling with time windows and scarce resources: temporal and resource-constrained project scheduling with regular and nonregular objective functions.* Springer-Verlag, Berlin, 2003.

[60] NICTA research team. NICTA research team. `http://nicta.com.au/research`, 2013.

[61] K. Nonobe and T. Ibaraki. A tabu search algorithm for a generalized resource constrained project scheduling problem. In *MIC2003: The fifth metaheuristics international conference*, pages 55–1, 2003.

[62] O. Ohrimenko, P. J. Stuckey, and M. Codish. Propagation via lazy clause generation. *Constraints*, 14(3):357–391, 2009. ISSN 1383-7133.

[63] online. Operations Management Group of the TU Clausthal. `http://www.wiwi.tu-clausthal.de/en/chairs/produktion/research/research-areas/project-generator/mrcpspmax/`, 2014.

[64] Operations Research and Scheduling research group of the University of Ghent. The multi-mode resource-constrained project scheduling problem. `http://www.projectmanagement.ugent.be/?q=research/project_scheduling/multi_mode`, 2011.

[65] A. A. B. Pritsker, L. J. Waiters, and P. M. Wolfe. Multiproject scheduling with limited resources: A zero-one programming approach. *Management Science*, 16 (1):93–108, 1969. doi: 10.1287/mnsc.16.1.93. URL `http://dx.doi.org/10.1287/mnsc.16.1.93`.

[66] F. Rossi, P. Van Beek, and T. Walsh. *Handbook of Constraint Programming.* Elsevier, 2006.

[67] A. Schnell. A hybrid local search algorithm integrating an exact CP-SAT approach to solve the multi-mode resource-constrained multi-project scheduling problem. In G. Kendall, G. Vanden Berghe, and B. McCollum, editors, *Proceedings of the 6th Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA)*, pages 802 – 806, Ghent, Belgium, August 2013.

[68] A. Schnell and R. F. Hartl. The effect of the predefined search space on recent exact algorithms for the resource-constrained project scheduling problem. *Working Paper*, 2012.

[69] A. Schnell and R. F. Hartl. On the generalization of constraint programming and boolean satisfiability solving techniques to schedule a resource-constrained project consisting of multi-mode jobs. *Working Paper*, 2013.

[70] A. Schnell and R. F. Hartl. On the efficient modeling and solution of the multi-mode resource-constrained project scheduling problem with generalized precedence relations. *Working Paper*, 2014.

[71] A. Schnell and R. F. Hartl. A new CP-SAT approach for the multi-mode resource-constrained project scheduling problem (MRCPSP). In Thomas Fliedner, Rainer Kolisch, and Anulark Naber, editors, *Proceedings of the 14th International Conference on Project Management and Scheduling*, pages 214–217, Munich, Germany, Mar 2014. TUM School of Management.

[72] J. Schulz. *Hybrid Solving Techniques for Project Scheduling Problems.* PhD thesis, Technische Universität Berlin, 2013.

[73] A. Schutt. *Improving scheduling by learning.* PhD thesis, Department of Computer Science and Software Engineering, The University of Melbourne, 2011.

[74] A. Schutt, T. Feydy, P. J. Stuckey, and M. G. Wallace. Explaining the cumulative propagator. *Constraints*, 16(3):250–282, 2011. ISSN 1383-7133. doi: 10.1007/s10601-010-9103-2. URL `http://dx.doi.org/10.1007/s10601-010-9103-2`.

[75] A. Schutt, G. Chu, P. J. Stuckey, and M. G. Wallace. Maximising the net present value for resource-constrained project scheduling. In Nicolas Beldiceanu, Narendra Jussien, and Eric Pinson, editors, *Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems*, volume 7298 of *Lecture Notes in Computer Science*, pages 362–378. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-29827-1.

[76] A. Schutt, T. Feydy, and P. J. Stuckey. Explaining Time-Table-Edge-Finding Propagation for the Cumulative Resource Constraint. In Carla Gomes and Meinolf Sellmann, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 7874 of *Lecture*

Notes in Computer Science, pages 234–250. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-38170-6.

[77] A. Schutt, T. Feydy, P. J. Stuckey, and M. G. Wallace. Solving RCPSP/max by lazy clause generation. *Journal of scheduling*, 16(3):273–289, 2013.

[78] C. Schwindt. *Generation of Resource Constrained Project Scheduling Problems Subject to Temporal Constraints*. Inst. für Wirtschaftstheorie und Operations-Research, 1998.

[79] A. Sprecher. Scheduling resource-constrained projects competitively at modest memory requirements. *Management Science*, 46(5):710–723, 2000. doi: 10.1287/mnsc.46.5.710.12044. URL `http://dx.doi.org/10.1287/mnsc.46.5.710.12044`.

[80] A. Sprecher. Network decomposition techniques for resource-constrained project scheduling. *Journal of the Operational Research Society*, 53(4):405–414, 2002.

[81] A. Sprecher and A. Drexl. Multi-mode resource-constrained project scheduling by a simple, general and powerful sequencing algorithm. *European Journal of Operational Research*, 107(2):431–450, 1998.

[82] A. Sprecher, S. Hartmann, and A. Drexl. An exact algorithm for project scheduling with multiple modes. *OR Spectrum*, 19(3):195–203, 1997.

[83] P. J. Stuckey, M. G. Banda, M. Maher, K. Marriott, J. Slaney, Z. Somogyi, M. Wallace, and T. Walsh. The G12 Project: Mapping Solver Independent Models to Efficient Solutions. In Maurizio Gabbrielli and Gopal Gupta, editors, *Logic Programming*, volume 3668 of *Lecture Notes in Computer Science*, pages 9–13. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-29208-1.

[84] P. Van Hentenryck. *Constraint Satisfaction in Logic Programming*. MIT press, 1989.

[85] P. Van Hentenryck and J.-P. Carillon. Generality versus Specificity: An Experience with AI and OR Techniques. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 660–664, 1988.

[86] V. Van Peteghem and M. Vanhoucke. Using resource scarceness characteristics to solve the multi-mode resource-constrained project scheduling problem. *Journal of Heuristics*, 17(6):705–728, 2011. ISSN 1381-1231. doi: 10.1007/s10732-010-9152-0. URL `http://dx.doi.org/10.1007/s10732-010-9152-0`.

[87] V. Van Peteghem and M. Vanhoucke. An experimental investigation of metaheuristics for the multi-mode resource-constrained project scheduling problem on new dataset instances. *European Journal of Operational Research*, 235 (1):62 – 72, 2014. ISSN 0377-2217. doi: http://dx.doi.org/10.1016/j.ejor.

2013.10.012. URL http://www.sciencedirect.com/science/article/pii/S0377221713008357.

[88] P. Vilím. Edge Finding Filtering Algorithm for Discrete Cumulative Resources in O(kn log(n)). In Ian P. Gent, editor, *Principles and Practice of Constraint Programming - CP 2009*, volume 5732 of *Lecture Notes in Computer Science*, pages 802–816. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-04243-0. doi: 10.1007/978-3-642-04244-7_62. URL http://dx.doi.org/10.1007/978-3-642-04244-7_62.

[89] P. Vilím. Timetable Edge Finding Filtering Algorithm for Discrete Cumulative Resources. In Tobias Achterberg and J. Christopher Beck, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 6697 of *Lecture Notes in Computer Science*, pages 230–245. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-21310-6. doi: 10.1007/978-3-642-21311-3_22. URL http://dx.doi.org/10.1007/978-3-642-21311-3_22.

[90] T. Wauters, J. Kinable, P. Smet, W. Vancroonenburg, G. Vanden Berghe, and J. Verstichel. The multi-mode resource-constrained multi-project scheduling problem. *Journal of Scheduling*, pages 1–13, 2014. ISSN 1094-6136. doi: 10.1007/s10951-014-0402-0. URL http://dx.doi.org/10.1007/s10951-014-0402-0.

[91] L. Zhang, C. F. Madigan, M. H. Moskewicz, and S. Malik. Efficient Conflict Driven Learning in a Boolean Satisfiability Solver. In *Proceedings of the 2001 IEEE/ACM International Conference on Computer-aided Design*, ICCAD '01, pages 279–285, Piscataway, NJ, USA, 2001. IEEE Press. ISBN 0-7803-7249-2. URL http://dl.acm.org/citation.cfm?id=603095.603153.

[92] G. Zhu, J.F. Bard, and G. Yu. A branch-and-cut procedure for the multimode resource-constrained project-scheduling problem. *INFORMS Journal on Computing*, 18(3):377–390, 2006.

List of Figures

2.1 Example: GPR between multi-mode jobs 10
2.2 AoN network for example 17
2.3 AoN network for the resulting SRCPSP 18
2.4 Visualisation of solution 19

3.1 Example 3.1.2, TP 32
3.2 Example 3.1.2, EF, Theorem 3.1.1 not applicable for job 1 32
3.3 Example 3.1.3: TP, compulsory parts 33
3.4 Example 3.1.3, EF, Theorem 3.1.1 applies for job 1 33
3.5 Example 3.1.3, EF, Lower bound update based on (3.14) 34
3.6 Clause learning based on the 1-UIP x_4 39
3.7 Deductions for non-chronological backtracking 40

4.1 AL preprocessing and parallel exact solution (ALPaPES) 50
4.2 Distribution of the Δ_k^{Best}-values (SCIP) 58
4.3 Distribution of Δ_k^{Best}-values (G12) 58
4.4 Distribution of Δ_k^{Best} for the new branching rule 59

List of Tables

4.1 Comparison on all 480 instances (SCIP) . . . 52
4.2 Comparison on all 480 instances (G12) . . . 52
4.3 Sol.time comparison on the non-easy instances . . . 53
4.4 Gap comparison on the open instances . . . 54
4.5 Comparison with recent exact approaches . . . 58
4.6 Comparison with recent exact approaches . . . 59
4.7 Results with the branching rule: Smallest variable index, smallest domain value . . . 59

5.1 Results on the 20-job instances . . . 73
5.2 Results on the 30-job instances . . . 74
5.3 Results on the 50-job instances . . . 74
5.4 Results on the 100-job instances . . . 74

6.1 30 Jobs: Results of our approaches . . . 86
6.2 30 Jobs: Comparison to `BABH` [39] w.r.t. `#opt` . . . 87
6.3 50 Jobs: Results of our approaches . . . 87
6.4 50 Jobs: Comparison to `BABH` [39] w.r.t. `#opt` . . . 88
6.5 100 Jobs: Results of our approaches . . . 89

7.1 Results for the instance set `A` . . . 94
7.2 Results for the instance set `B` . . . 95

List of Algorithms

1 Preprocessing: Elimination of modes and nonrenewable resources . . . 16

2 Backtracking with constraint propagation 24
3 BaB with constraint propagation 27

Abstract

In practice, project managers are confronted with the problem of assigning starting times to a number of precedence-related project milestones (activities) so that a certain project deadline is respected. When the amount of available resources is limited and the number of activities is large, this is a highly complex task. In the scientific literature, this issue is known as the resource-constraint project scheduling problem (RCPSP). Many variants of the RCPSP introducing various problem characteristics and optimization goals have been considered in the literature.

In this dissertation, the main research topic is the analysis of new exact solution approaches for the single-mode RCPSP (SRCPSP) and the multi-mode RCPSP (MRCPSP) with standard precedence relations (SPRs) and generalized precedence relations (GPRs). More precisely, we only consider solution approaches which provide the proven optimal solution of the problem at hand on termination. Exact algorithms are of high relevance when it comes to evaluate the quality of solutions determined through heuristics.

In the SRCPSP, every job can be processed in one predefined mode. The MRCPSP is a generalization of the SRCPSP where every activity can be processed in multiple modes with varying durations and resource consumptions, i.e. the mode assignment also becomes part of the problem. If one job can only start after another job is finished, the latter two jobs are in a SPR. GPRs arise if the minimum gap between the starting times of two precedence related jobs can be arbitrary (i.e. also negative). In this situation, also cases can be modeled where one job has to start before the end of another job. In general, the occurrence of GPRs makes the RCPSP more complicated.

Recently, Branch and Bound (BaB) algorithms combining Constraint Programming (CP) and Boolean Satisfiability (SAT) Solving techniques have been highly successful for variants of the SRCPSP. As a first contribution, we analyze these solution approaches w.r.t. performance variability. We vary certain initial conditions to point out a significant erraticism in the state-of-the-art CP-SAT approaches for the SRCPSP with SPRs. Moreover, a parallel solution approach is proposed which takes advantages of this performance variability.

The main contribution is an extension of the success story of the CP-SAT approaches for the SRCPSP to the MRCPSP with SPRs and GPRs. We provide and compare various new models of the latter problems in two optimization frame-

works, which support the solution by a BaB algorithm integrating CP and SAT techniques. A majority of the models integrate new constraint handlers, namely `cumulativemm`, `sprecedencemm` and `gprecedencemm`, which we implemented for the optimization framework SCIP. These constraints offer the possibility to efficiently formulate renewable resource constraints, SPRs and GPRs in the context of multi-mode jobs. Moreover, they capture constraint propagation and explanation generation algorithms to strengthen the overall solution procedure.

Our best performing solution approaches significantly outperform the state-of-the-art exact algorithms for these MRCPSP classes on certain benchmark sets from the literature. Furthermore, we close (find the optimal solution and prove its optimality for) 324 open 50- and 100-job instances of the MRCPSP with SPRs from the literature. In addition, we improve the best known makespan for 176 instances from the latter benchmark set. For the MRCPSP with GPRs, 289 open instances with 30, 50 and 100 jobs can be closed through our CP-SAT approaches and we find better feasible solutions for 271 of these instances. In total, the obtained results are highly promising.

The dissertation ends with an extensive proposal of future research directions in the area of CP-SAT approaches for scheduling problems.

Zusammenfassung

In der Praxis sind Projekt Manager oft mit dem Problem konfrontiert, den in Zusammenhang stehenden Meilensteinen eines Projekts (auch Vorgänge genannt), Startzeitpunkte zuzuweisen, sodass eine termingerechte Fertigstellung des Projekts gewährleistet ist. Wenn die verfügbaren Ressourcen beschränkt sind und das Projekt aus einer großen Anzahl von Vorgängen besteht, ist dies eine schwierige Aufgabe. In der wissenschaftlichen Literatur wird diese Planungsaufgabe das ressourcenbeschränkte Projektplanungsproblem (RCPSP) genannt. Eine Vielzahl von Varianten des RCPSPs mit unterschiedlichen Problemcharakteristiken und Optimierungszielen wurde bereits von Forschern betrachtet.

In dieser Dissertation geht es hauptsächlich um eine Untersuchung von neuen exakten Algorithmen für das RCPSP mit einer und mehreren Ausführungsalternativen (SRCPSP und MRCPSP) und mit Vorrangbeziehungen und allgemeinen Zeitbeziehungen. Wir betrachten hier Verfahren, deren Ziel es ist, nicht nur eine möglichst gute Lösung zu finden, sondern auch die Optimalität der besten gefundenen Lösung zu beweisen. Die Weiterentwicklung von exakten Algorithmen ist von großer Bedeutung, da sie ein Qualitätsmaß für die Bewertung von heuristischen Lösungen liefern.

Beim SRCPSP kann jeder Vorgang nur in einer fest vorgegebenen Variante durchgeführt werden. Wenn die Vorgänge in mehreren Varianten mit unterschiedlichen Bearbeitungszeiten und einem unterschiedlichen Ressourcenverbrauch durchgeführt werden können, spricht man von dem MRCPSP. Hier stellt sich also auch die Frage der Bestimmung einer Ausführungsalternative. Außerdem werden Problemklassen mit Vorrangbeziehungen und allgemeinen Zeitbeziehungen betrachtet. Wenn ein Vorgang erst nach Ende eines anderen Vorgangs beginnen kann, bezeichnet man dies als Vorrangbeziehung zwischen den beiden Vorgängen. Wenn die zu berücksichtigenden Zeitabstände zwischen den Startzeitpunkten zweier in Zusammenhang stehender Vorgänge beliebig, also möglicherweise auch negativ, sein können, spricht man von allgemeinen Zeitbeziehungen. Hierbei können auch Fälle modelliert werden, bei denen ein Vorgang vor dem Ende eines anderen Vorgangs beginnen muss. Im Allgemeinen steigt die Komplexität des RCPSP, wenn allgemeine Zeitbeziehungen auftreten.

Erst kürzlich hat sich herausgestellt, dass Branch and Bound-Verfahren, die Prinzipien aus der Constraintprogrammierung (CP) und dem SAT-Solving integri-

eren, sehr gute Ergebnisse bei der Lösung von Varianten des SRCPSP erzielen. Der erste Forschungsbeitrag dieser Dissertation beinhaltet eine Analyse dieser Verfahren in Bezug auf Unregelmäßigkeiten im Lösungsverlauf. Wir variieren hierzu spezielle Anfangszustände des Problems um diese Unregelmäßigkeiten zu veranschaulichen. Außerdem wird ein Lösungsansatz vorgeschlagen, der auf Konzepten der parallelen Programmierung basiert, um diese Unregelmäßigkeiten auszunutzen.

Der hauptsächliche Forschungsbeitrag dieser Dissertation besteht in der Anwendung und Erweiterung der bestehenden CP-SAT Verfahren, sodass Instanzen des MRCPSP mit Vorrangbeziehungen und allgemeinen Zeitbeziehungen in effizienter Weise exakt gelöst werden können. Hierzu vergleichen wir zahlreiche neue Modelle des MRCPSP in zwei Optimierungs-Frameworks, die einen Lösungsansatz bestehend aus CP und SAT-Solving Techniken bereitstellen. Der Großteil dieser Modelle verwendet neue globale Constraints mit der Bezeichnung `cumulativemm`, `sprecedencemm` und `gprecedencemm`, die wir für das Optimierungs-Framework SCIP implementiert haben. Diese Konstrukte bieten die Möglichkeit Nebenbedingungen zu erneuerbaren Ressourcen, Vorrangbeziehungen und allgemeinen Zeitbeziehungen im Kontext von mehreren Ausführungsalternativen in effizienter Weise mathematisch zu modellieren. Überdies integrieren sie problemspezifische Propagierungsalgorithmen und Schnittstellen zum internen SAT-Solving Mechanismus von SCIP. Dies dient der Verbesserung des Standardlösungsverfahrens von SCIP.

Für spezielle Instanzen des MRCPSP mit Vorrangbeziehungen und allgemeinen Zeitbeziehungen aus der Literatur erzielen wir mit unseren besten Modellen und zugehörigen Lösungsverfahren bessere Ergebnisse als die besten bekannten exakten Verfahren für diese Problemstellungen. Wir konnten mit unseren Lösungsverfahren 324 offene Instanzen des MRCPSP mit Vorrangbeziehungen schließen. Unser bestes Verfahren konnte also für diese Instanzen die zuvor nicht bekannte optimale Lösung finden und die Optimalität derselben beweisen. Für 176 Instanzen des MRCPSP mit Vorrangbeziehungen aus der Literatur konnten wir die beste bekannte Gesamtprojektdauer verbessern. Des Weiteren konnten wir 289 offene Instanzen des MRCPSP mit allgemeinen Zeitbeziehungen schließen und die beste bekannte Gesamtprojektdauer für 271 Instanzen verbessern. Insgesamt sind die erzielten Resultate sehr vielversprechend.

Zum Abschluss der Dissertation liefern wir eine Vielzahl von Vorschlägen für zukünftige Forschungsthemen im Bereich von CP-SAT Ansätzen für Projektplanungsprobleme.

Curriculum Vitæ

Education

2010 – now, ongoing **Doctor of Philosophy**, *University of Vienna, Department of Business Administration*, Austria.
Participation in the PhD-program "Logistics and Operations Management", Supervisor: Richard F. Hartl

2007 – 2010 **Master of Science**, *University of Salzburg, Faculty of Natural Sciences*, Salzburg, *passed with distinction*.
Master Study of General Mathematics.

2004 – 2007 **Bachelor of Science**, *University of Salzburg, Faculty of Natural Sciences*, Salzburg, *passed with distinction*.
Study of Mathematics.

1994 – 2003 **Abitur**, *Karlsgymnasium Bad Reichenhall, secondary school*.

Master thesis

Title *Solving the districting problem by integer programming techniques (in german)*

Supervisors Prof. Wolfgang Schmid, Dr. Günter Kiechle

Description Detailed theoretical description of concepts from integer programming and an application of the latter to a districting problem in the state of Salzburg.

Vocational Experience

University

August 2010 – now **Research Assistant**, *University of Vienna*, Austria.
Teaching, administration tasks and research for PhD-thesis:
- An Application of Constraint Programming and Boolean Satisfiability Solving Techniques to Variants of the Resource-Constrained Project Scheduling Problem

Research-oriented internships

July 2009 – May 2010 **Internship|Master student**, *Salzburg Research Forschungsgesellschaft mbH*, Salzburg.
Master thesis project:
- Real-world application: Analysis and optimization of the service territory design of an industrial partner in the state of Salzburg, Austria.

2007 – 2008 **Internship**, *Brückner Maschinenbau (Mechanical Engineering) GmbH*, Siegsdorf.

Employment in the Department of Research & Development:

- Elaboration of mathematical concepts for the measurement of the biaxial birefringence of optical films
- Programming the synchronization of a positioning system and the measurements of a birefringence instrument to automatize the birefringence measurement procedure.

Other

2003 – 2004 **Civilian service**, *Transport service at the Bavarian Red Cross in Bad Reichenhall*.

Academical Experience

Teaching Experience

October 2011 – now Bachelor Course "Production and Logistics" (in german):

Supervising Experience

May 2012 – now Supervision of various bachelor theses and co-supervision of two master theses with the following topics:

- "Adaptive Large Neighborhood Search for the Curriculum-Based Timetabling Problem", finished in December 2013
- "A comparison of MILP models for the multi-mode resource constrained project scheduling problem", ongoing.

Reviewing Experience

European Journal of Operational Research, Central European Journal of Operations Research, ORP3 2012 Conference Papers

Competitions

February 2013 – August 2013 Participation in the competition of the Multidisciplinary International Scheduling Conference (MISTA):

- Optimization of instances of the multi-project multi-mode RCPSP with the aim of total project delay and makespan minimization
- **7th**-rank in the qualification round
- **9th**-rank in the final round.

Prizes

April 2014 14th International Conference on Project Management and Scheduling:

- Submission awarded with the third prize in the Best Student Paper Competition (See [2] and [10]).

Publications

Conference Presentations

[1] Richard F. Hartl Alexander Schnell. On the efficient modeling of the MRCPSP with GPRs in the optimization framework SCIP. In *International Conference on Operations Research*, Aachen, Germany, September 2014.

[2] Richard F. Hartl Alexander Schnell. A new CP-SAT approach for the multi-mode resource-constrained project scheduling problem (MRCPSP). In *14th International Conference on Project Management and Scheduling*, Munich, Germany, March-April 2014.

[3] Richard F. Hartl Alexander Schnell. On the extension of recent CP-SAT approaches from single-mode to multi-mode jobs in RCPS. In *2nd International Optimisation Summer School*, Kioloa, Australia, January 2014.

[4] Richard F. Hartl Alexander Schnell. A new exact approach for the multi-mode resource-constrained project scheduling problem (MRCPSP). In *Young Academic's Management Science (YAMS)*, Salzburg, Austria, December 2013.

[5] Richard F. Hartl Alexander Schnell. A combined CP-SAT-MIP approach for the multi-mode resource-constrained project scheduling problem. In *International Conference on Operations Research (GOR)*, Rotterdam, Netherlands, September 2013.

[6] Alexander Schnell. A hybrid local search algorithm integrating an exact CP-SAT approach to solve the multi-mode resource-constrained multi-project scheduling problem. In *Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA)*, Ghent, Belgium, August 2013.

[7] Alexander Schnell. The impact of the predefined search space on recent exact algorithms for the RCPSP. In *21st International Symposium on Mathematical Programming*, Berlin, Germany, August 2012.

[8] Alexander Schnell and Richard F. Hartl. A preprocessing procedure to improve recent exact algorithms for the resource-constrained project scheduling problem. In *25th European Conference on Operational Research*, Vilnius, Lithuania, July 2012.

[9] Alexander Schnell. A generalized forward-backward improvement heuristic for the resource-constrained project scheduling problem. In *2nd Workshop on Young Academics' Management Science*, Graz, Austria, December 2011.

Working Papers and Publications

[10] Alexander Schnell and Richard F. Hartl. A new CP-SAT approach for the multi-mode resource-constrained project scheduling problem (MRCPSP). In Thomas Fliedner, Rainer Kolisch, and Anulark Naber, editors, *Proceedings of the 14th International Conference on Project Management and Scheduling*, pages 214–217, Munich, Germany, Mar 2014. TUM School of Management.

[11] Alexander Schnell. A hybrid local search algorithm integrating an exact CP-SAT approach to solve the multi-mode resource-constrained multi-project scheduling problem. In *Proceedings of the Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA)*, Ghent, Belgium, August 2013.

[12] Alexander Schnell and Richard F. Hartl. On the efficient modeling and solution of the multi-mode resource-constrained project scheduling problem with generalized precedence relations. *Working Paper, submitted for publication*, 2014.

[13] Alexander Schnell and Richard F. Hartl. On the generalization of constraint programming and boolean satisfiability solving techniques to schedule a resource-constrained project consisting of multi-mode jobs. *Working Paper, submitted for publication*, 2014.

[14] Alexander Schnell and Richard F. Hartl. The effect of the predefined search space on recent exact algorithms for the resource-constrained project scheduling problem. *Working Paper*, 2012.