Hybrid Solving Techniques for Project Scheduling Problems

vorgelegt von
Diplom-Wirtschaftsmathematiker
Jens Schulz
Berlin

Von der Fakultät II – Mathematik und Naturwissenschaften
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften
Dr. rer. nat.

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender:	Prof. Dr. Martin Keller-Ressel
Berichter:	Prof. Dr. Rolf H. Möhring
Berichter:	Prof. Dr. Marco E. Lübbecke

Tag der wissenschaftlichen Aussprache: 14.12.2012

Berlin 2013

D 83

Contents

Introduction **2**

1 Basics: Scheduling and solving techniques **6**

1.1 RCPSP 6
1.1.1 Problem definition 6
1.1.2 Complexity status 7
1.2 Notation for scheduling problems 9
1.2.1 Machine scheduling 9
1.2.2 Project scheduling 10
1.3 Solving techniques 10
1.3.1 Mixed Integer Programming 11
1.3.2 Constraint Programming 12
1.3.3 Satisfiability testing (SAT) 15
1.3.4 Constraint Integer Programming 16
1.4 Related work 18
1.5 Benchmark instances and computational studies 21
1.5.1 Benchmark instances and instance indicators 21
1.5.2 Computational studies 23
1.5.3 Setup for experiments 24

2 CIP Techniques for RCPSP **27**

2.1 CP techniques 28
2.1.1 Constraint propagation 29
2.1.2 Propagation of basic constraints 30
2.1.3 The cumulative constraint 33
2.1.4 An approximative criterion for energetic reasoning 41
2.2 Hybrids of CP and SAT 49
2.2.1 Conflict analysis on integer variables 49
2.2.2 Explanations for the basic constraints 53
2.3 Explanations for the cumulative constraint 54
2.3.1 Complexity of delivering optimal explanations 54
2.3.2 Explanation algorithms 60
2.3.3 Computational study 62
2.4 A continuous relaxation of the cumulative constraint 77
2.4.1 Separation procedure and an example 79
2.4.2 Relaxations and bounding techniques 80
2.4.3 Computational study 86

3 Presolving and branching schemes for RCPSP **91**
3.1 Lower bounds . . . 92
3.1.1 Known lower bounds . . . 92
3.1.2 A preemptive lower bound . . . 94
3.2 Presolving techniques . . . 96
3.2.1 Coefficient strengthening . . . 97
3.2.2 Redundant resources . . . 100
3.2.3 Strengthening transitive precedence constraints . . . 102
3.2.4 Computational study . . . 104
3.3 Tree search algorithms . . . 110
3.3.1 Branching schemes . . . 111
3.3.2 Computational study . . . 114
3.3.3 Concluding discussion . . . 121

4 Applications **123**
4.1 Application to RCPSPDC . . . 123
4.1.1 Problem description . . . 124
4.1.2 Related work . . . 125
4.1.3 Solution approach . . . 127
4.1.4 Computational study . . . 130
4.2 Application to Labor-Constrained Scheduling Problems . . . 135
4.2.1 Problem description . . . 135
4.2.2 Related work . . . 136
4.2.3 The `labor`-constraint . . . 136
4.2.4 Computational study . . . 143

5 Turnaround Scheduling **154**
5.1 Problem description and related work . . . 155
5.2 Turnaround heuristics . . . 161
5.3 Mixed Integer Programming formulations . . . 162
5.3.1 Obstacles of MIP formulations . . . 163
5.3.2 Exponential formulation . . . 164
5.3.3 Branch-price-and-cut algorithm . . . 167
5.4 Computational study . . . 170
5.4.1 Benchmark instances . . . 170
5.4.2 Results . . . 171

Bibliography **178**

Introduction

Scheduling problems are among the most widely studied problems in computer science, mathematics and operations research. Scheduling decisions are involved in a huge amount of decision and optimization problems arising in practice, for instance, in computer systems as well as in tactical, strategical and operational production planning. Due to the practical relevance and in many cases due to the intrinsic hardness, these problems have challenged researchers with theoretical as well as with practical background.

In a *scheduling* problem, we want to find start times for a set of jobs that have certain characteristics with respect to a set of side constraints, such as precedence relations or resource constraints. These problems arise in many applications, such as setting up time-tables in schools and universities, in production planning and when assigning crews to specific operations. Due to these and numerous other applications, scheduling problems have been widely studied and various optimization techniques have been developed to solve problems of this type. These problems often contain different characteristics that make these problems already $\mathcal{NP}$-hard. To compute optimal solutions to such problems is a challenging task. Different communities from the OR-society, such as Integer Programming (IP), Constraint Programming (CP) and Satisfiability Solving (SAT) developed exact solvers with different abilities to model and solve optimization problems.

MIP solving is restricted to problems that can be modeled via binary, integer or continuous decision variables coupled with linear constraints. An LP relaxation is solved in each node of the branch-and-bound tree with the heavy use of cutting plane algorithms in order to increase the dual bound and thereby prune unpromising search spaces. Further techniques, such as Lagrangean relaxation, Dantzig-Wolfe decomposition or Benders decomposition can be used to cope with large instances. In a CP solver where usually no objective function is present, propagation algorithms are applied in each node of the search tree in order to shrink the domains of the variables until a feasible solution is found if the problem is not infeasible at all. In SAT solvers, problems are model via binary variables (true or false) and clauses over these variables in conjunctive normal-form. A technique called conflict analysis is used to speed up the search. Thereby, infeasible states are analyzed in order to perform backtracking or to learn new valid clauses.

The goal of this thesis is to develop and compare solution techniques in a hybrid framework that integrates the different techniques from mathematical programming (CP, IP and SAT) in one search tree. In particular, we study the complexity of delivering optimal explanations for propagation algorithms and the complexity of deriving lower bounds through a continuous relaxation, both for cumulative scheduling problems, such as the very general Resource-Constrained Project Scheduling Problem (RCPSP). The techniques developed for this problem will also be applied to related problems, such as RCPSP with discounted cashflows, a labor-constrained scheduling problem and a resource leveling

problem as it occurs, e.g., in chemical manufacturing. Heading for a generic solver, we show how the techniques are best integrated and measure the impact of preprocessing, and of different propagation algorithms, relaxations and branching rules.

Main lessons. RCPSP is a fascinatingly complex problem which has attracted many researchers over the last decades. It is widely believed that CP techniques perform best on these instances as they are of high logical structure that can be used throughout search. This thesis sheds some light on the strengths of the different techniques from IP, CP and SAT. We show the merits of using conflict analysis as well as IP techniques to tackle such problems.

We will show that it pays off to compute explanations of minimum size as this creates smaller conflict clauses. Another important strength of our solver is the use of generic branching rules. In contrast to problem specific branching rules, learning from infeasible states is important to guide the search by branching on variables that are often involved in conflicts. Guiding search by pseudocost as done in MIP solving, is even better than search schemes from CP and SAT. Using the logical structure and identifying easy as well as hard parts of the instances is important to handle the hard scheduling instances. We propose to transfer coefficient strengthening techniques from MIP to CP's global cumulative constraint in order to derive better bound adjustments. Similarly, the detection of subnetworks that induce their own lower bounds on transitive precedence relations is a key ingredient to improve the dual bounds of several instances.

The application of these hybrid techniques to net present value problems with a more complex objective function shows that using the proposed continuous relaxation improves over a poor CP approach by far, if the objective function is complex with positive and negative coefficients. On the other hand, our results on the Pack instances for RCPSP and the application to LCSP show that a pure CP approach performs well if it is hard to generate good conflicts. Strong dual bounds by IP, e.g., by Dantzig-Wolfe decomposition, can be a key technique to solve these problems and in particular to close the optimality gap if the heuristics are already doing well.

Outline of the thesis

Chapter 1: We start by introducing the very general RCPSP which serves as the basic scheduling problem in our studies. Second, the solving methodologies Integer Programming, Constraint Programming and Satisfiability solving are introduced. These techniques will be applied to scheduling problems in a hybrid framework, called Constraint Integer Programming (CIP). Benchmark instances, instance indicators and results from other works are presented to make the reader familiar with RCPSP and its characteristics.

Chapter 2: Due to the logical implications induced by the resource constraints, CP techniques work well to solve even large RCPSP instances. Our first computational study on a close integration of IP, CP and SAT techniques has been published in [29] which shows that IP and SAT techniques are valuable add-ons to the CP part depending on the instance characteristics and the size of the instance.

Section 2.1 starts with an overview of the main CP propagation algorithms for the `cumulative` constraint, such as time-tabling, edge-finding, time-table edge-finding and energetic reasoning. As computational results reveal, on most instances it is best to

use time-tabling as the only propagation algorithm due to faster running times. A lot of work has been carried out to improve the energy-based propagation algorithms, such as energetic reasoning. For example, Kooli et.al. [165] propose to solve MIPs in order to detect intervals with high energy consumption and thereby to detect infeasible nodes early. On the contrary, in Section 2.1.4 we propose to identify promising intervals by eagerly smearing the requested energy of a job over its processing window. The resulting energy estimate does not underestimate the true energy contribution by more than a factor of three. We embed our approximative criterion into the energetic reasoning propagation algorithm and perform a computational study which reveals that on PSPLib instances the estimated energy contribution is close to the true energy demand. This way, we successfully restrict the number of intervals per node to be considered and obtain a speed-up factor of four. This is joint work with Timo Berthold and Stefan Heinz from the Zuse Institute Berlin [30].

The most important technique that comes from SAT is to learn from infeasible search states. Here, *learning* results in additional constraints which induce further propagation. Besides that, branching scores on the variables are kept which indicate which variables are involved in many conflicts. When analyzing infeasible states, we need to explain which variables invoked which bound change. For several types of constraints, such as logicor, set-partitioning, and so on these explanations are unique. This is not the case for some propagation algorithms of the cumulative constraint. We introduce the notion of *explanation algorithms* that seek to find *optimal explanations* for the bound adjustments of these propagation algorithms, see Section 2.2. Initial experiments have been published in [137] in cooperation with Stefan Heinz.

From the IP world different exact formulations have been developed. These either suffer from slow solving times on large instances or from poor lower bounds. In Section 2.4, we develop a new continuous relaxation of the cumulative constraint that is not exact but enables our solver to use lower bounds from LP, to perform branching according to the LP values and most important to collect pseudocost values to guide the choice of the branching variables.

Chapter 3: A huge amount of RCPSP instances which contain between 30 and 60 jobs cannot be solved to optimality nowadays. The huge duality gap of about 60% between a critical path lower bound and initial solutions from simple list scheduling heuristics show that stronger lower bounds on the makespan are highly demanded to come up with competitive exact approaches.

In Section 3.2, we devise two approaches in order to tighten the lower bounds. First, we generalize coefficient strengthening techniques from MIP to CP's cumulative constraint. This helps to close 10 additional Pack instances, as this way the energy-based propagation algorithms are able to detect stronger bound adjustments. Second, we study transitive precedence relations, where the distance between such two jobs can be strengthened by volume and disjunctive arguments of the induced subnetworks. We propose a preemptive propagation algorithm to compute such lower bounds.

The applied branching scheme has a huge impact on the solving process. We show in Section 3.3 that generic branching schemes perform much better in a CIP framework on disjunctive instances from standard benchmark libraries, in contrast to problem-based branching schemes, which in return perform well on highly cumulative instances.

At the end of this chapter, we compare the CP, IP and SAT techniques with each

other and to other approaches. The results show that applying conflict analysis from SAT remarkably helps to prune unpromising nodes. The numbers further indicate that the proposed continuous relaxation helps to make good branching decisions based on pseudocost scores. In particular, highly cumulative instances with low makespan are best solved via IP approaches, whereas on the disjunctive instances a CP-SAT approach performs best.

Chapter 4: RCPSP has a simple objective function which contains only one variable, the makespan variable, with coefficient one. More complex cost functions such as exponential functions are part of many practical applications. In Section 4.1, we apply our techniques to RCPSP with the objective to maximize net present value, i.e., the sum over all cash-in minus cash-out flows. We show that a CP-SAT hybrid performs best on instances where all coefficients are either positive or negative. If both kinds of coefficients occur, the use of the continuous relaxation from Chapter 2 becomes a powerful add-on. With this CIP approach on inhomogeneous instances more best upper and lower bounds can be derived in contrast to a pure CP-SAT hybrid.

In a second case study (Section 4.2), the CIP solving techniques for RCPSP are applied to a labor-constrained scheduling problem in which the resource demands per job vary over time. On such kinds of instances it turns out that a pure CP approach performs best with respect to primal and dual bounds. One instance from a given benchmark set can be closed with our approach, while we are only able to improve primal bounds in contrast to pure CP approaches. Tabu search heuristics from the literature report better results there.

Chapter 5: In this chapter we study the *Turnaround Scheduling* problem as it occurs in the shut-down of chemical plants. We show for a complex subproblem, the resource leveling problem, that a Dantzig-Wolfe reformulation yields optimal solutions within reasonable running times while performing much better than a standard MIP solver. The computational study reveals that the dual bounds after reformulation are much stronger and therefore yield the optimality proof. In this context, we transform well-known strong precedence inequalities to the reformulated problem and study how to best separate them as cuts throughout search. It turns out that with these cuts the number of nodes remarkably decreases while unfortunately also the solving times increase. Anyway, few hard instances can be solved faster using these cuts.

The studied problem and the heuristics have been introduced in Megow et al. [183], while the branch-price-and-cut framework is joint work with E.T. Coughlan and M.E. Lübbecke. Preliminary results have been published in [67].

Chapter 1

Basics: Scheduling and solving techniques

Resource-constrained scheduling problems have been widely studied and different solving paradigms such as Constraint Programming, Satisfiability Solving, Integer Programming and heuristics have been applied to solve these problems. In this chapter, we introduce the problem and the most important frameworks and results from the literature. These build the fundament of our study in which hybrid techniques are developed and closely integrated in one search tree.

In Section 1.1, we formally introduce RCPSP that includes several scheduling problems in one model. The notation that is used throughout this thesis is introduced in Section 1.2. In Section 1.3, we present the three basic techniques and a hybridization of them, *Constraint Integer Programming*, to tackle optimization problems. We conclude by presenting studies from the literature in Section 1.5.2 where several approaches are compared to each other.

1.1 RCPSP

1.1.1 Problem definition

The Resource-Constrained Project Scheduling Problem (RCPSP) is one of the most widely studied problems in the scheduling community. This is due to its practical relevance and its inherent computational complexity, see Section 1.1.2.

In RCPSP we are given a set $\mathcal{J}$ of n non-preemptable jobs and a set $\mathcal{R}$ of renewable resources. Each resource $k \in \mathcal{R}$ has bounded capacity $R_k \in \mathbb{N}$. Every job $j \in \mathcal{J}$ has a processing time $p_j \in \mathbb{N}_0$ and resource demands $r_{jk} \in \mathbb{N}_0$ of each resource $k \in \mathcal{R}$. W.l.o.g., we assume that all these parameters are integer. A *schedule* $\boldsymbol{S} \in \mathbb{N}_0^n$ is an assignment of integer start times S_j for each job j. The start time S_j of job j is constrained by its predecessors that are given by a precedence graph $G = (V, E)$. In the node set V there is exactly one node j for each job j. A directed edge $(i, j) \in E$ represents a precedence relationship, i.e., job i must be finished before job j starts. This can be expressed by the inequality $S_i + p_i \leq S_j$. Schedules that obey all precedence relations are called *precedence-feasible*. We assume that a first and a last dummy job with zero processing times exist which model the start and end of the project. Their start times are denoted by S_0 and S_{n+1}. All other jobs succeed job 0 and proceed job $n + 1$. The

start time S_{n+1} of the last dummy job is also denoted by $C_{\max}$, the *maximum completion time*: $C_{\max} = \max_{j \in \mathcal{J}}\{S_j + p_j\}$.

Another set of constraints are the *resource constraints* which require for each point in time that the cumulated demand of all jobs running at that point in time, must not exceed the given capacity:

$$\sum_{j:S_j \leq t < S_j + p_j} r_{jk} \leq R_k \qquad \forall\, t, \forall\, k \in \mathcal{R}. \tag{1.1}$$

If a schedule respects all resource constraints, it is called *resource-feasible.* Finally, a schedule is called *feasible* if it is precedence- and resource-feasible.

The goal in RCPSP is to find a feasible schedule such that the latest completion time of all jobs, the *makespan*, is minimized. The problem formulated as a non-linear program reads as follows:

$$\begin{aligned}
\min \quad & C_{\max} && && (1.2)\\
\text{subject to} \quad & S_i + p_i \leq S_j && \forall\,(i,j) \in E && (1.3)\\
& \sum_{j:S_j \leq t < S_j + p_j} r_{jk} \leq R_k && \forall\, t, \forall\, k \in \mathcal{R} && (1.4)\\
& S_j \in \mathbb{N}_0 && \forall\, j \in \mathcal{J}. && (1.5)
\end{aligned}$$

Example 1.1. Figure 1.1 shows a precedence network with seven jobs and a possible schedule that respects the resource and precedence constraints. A schedule is given by a gantt chart, where the horizontal axis indicates the time and the vertical axis the resource usage. In Figure 1.1, the resource capacities are given by $R_1 = 3$ and $R_2 = 4$. The processing times and resource demands are given as follows:

j	A	B	C	D	E	F	G
p_j	3	2	2	4	2	4	3
r_{j1}	2	2	0	0	2	0	1
r_{j2}	1	0	2	3	0	1	3

1.1.2 Complexity status

RCPSP is strongly $\mathcal{NP}$-hard and this hardness stems from different problem characteristics, such as Knapsack or Job-Shop-Scheduling. Garey and Johnson [116] show that a very simple special case of RCPSP makes the problem already intractable. They show strong $\mathcal{NP}$-hardness by a reduction from 3-Partition. In this case, RCPSP consists of one resource with capacity 3, no precedence constraints and each job has a processing time of one.

Ullman [251] shows by reduction from 3-SAT that Precedence Constrained Scheduling is $\mathcal{NP}$-complete already for a deadline of three. This problem asks whether a feasible schedule exists, given a set of jobs with unit processing time, unit resource demands, precedence constraints, a single resource with capacity of $m \geq 3$ and an overall deadline. But here, the capacity m can get arbitrarily large. Additionally, Ullman shows that the case $m = 2$ is $\mathcal{NP}$-complete if processing times are selected from $\{1, 2\}$.

Blazewicz et al. [33] show that many machine scheduling problems with the objective to minimize the makespan become strongly $\mathcal{NP}$-hard as soon as resource constraints

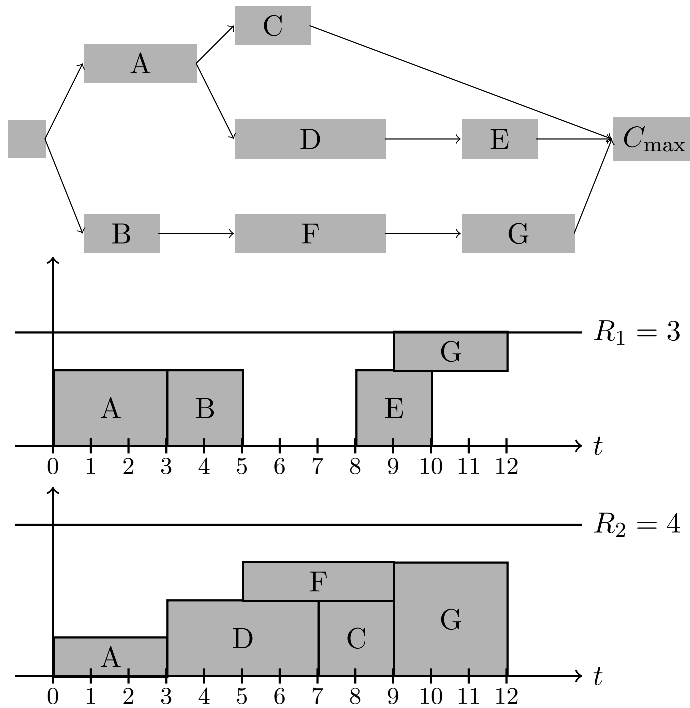


Figure 1.1: A precedence network of seven jobs and two dummy jobs (start and end) and an optimal solution with makespan $C_{\max} = 12$ are depicted.

are part of the model. They provide a stronger result than Ullman, as they show in Theorem 7 [33] strong $\mathcal{NP}$-hardness for makespan minimization on two parallel machines, one resource with capacity one, unit processing times and even if the precedence relations correspond to chains.

Furthermore, RCPSP is strongly $\mathcal{NP}$-hard by reduction from Bin-Packing – it generalizes Bin-Packing to multiple dimensions. In this generalization, each resource constraint models one dimension, jobs have unit processing time and the goal is to minimize the makespan, see Garey et al. [192]. Reduction works again from 3-Partition as in [116].

Furthermore, RCPSP models the famous graph coloring problem and is therefore strongly $\mathcal{NP}$-hard, as shown by Schäffter [221] and later Blazewicz et al. [33]. In graph coloring the goal is to minimize the chromatic number, i.e., the number of colors that suffices to color all the vertices such that two adjacent vertices have different colors. For this reduction unit processing times, unit demands and exactly one disjunctive resource per edge from the coloring graph are needed. No precedence constraints are introduced. This reduction induces bad approximability results for RCPSP, see [110]. They show that unless $\mathcal{P} \subseteq \mathcal{ZPP}$, or equivalently $\mathcal{NP} \not\subseteq co\mathcal{NP}$ it is intractable to approximate the chromatic number within $n^{1-\epsilon}$ for any $\epsilon > 0$, where n is the number of vertices of the graph. In the reduction, one disjunctive resource per edge in the graph is needed. This hardness result is of practical relevance, because in RCPSP a lot of jobs are disjunctive due to the resource demands from different cumulative resource constraints. Hence, this reduction underlines the hardness of disjunctive RCPSP instances.

The more recent results by Gafarov et al. [115] also underline the hardness of RCPSP. The authors show that for a special case with one resource constraint no polynomial-time constant factor approximation algorithm exists. Furthermore, they prove that the

preemptive relaxation yields a ratio of $O(n \log(n))$.

1.2 Notation for scheduling problems

We distinguish two classes of scheduling problems: *machine scheduling problems* and *project scheduling problems*. In the former problem, machine assignments for a set of jobs need to be found. In the latter problem, start times need to be assigned for each job subject to resource and precedence constraints. Mixtures and numerous variations of these problems exist.

As various communities treated such problems, conflicting notations have been developed. In project scheduling r_j symbolizes the resource demand while earliest start times (release dates) are denoted by est_j, whereas in machine scheduling, the release dates are denoted by r_j. We mainly focus on project scheduling problems and stay coherent with this notation as introduced by Brucker et al. [36]. Resource demands per job j are denoted by r_j and release dates (earliest start times) by est_j. When considering relations to machine scheduling problems, we change the font of the parameters and write $\mathsf{r_j}$ for release dates and $\mathsf{size_j}$ for resource demands. Both notations are introduced in the next sections.

1.2.1 Machine scheduling

The $\alpha|\beta|\gamma$-classification scheme [127] constitutes a precise and compact formulation for general machine scheduling problems. It is used throughout the literature and we will relate our work (lower bounds by a continuous relaxation of the cumulative constraint, see Section 2.4) to well established results.

In the $\alpha|\beta|\gamma$-scheme, parameter α models the machine environment, β the job characteristics and γ the optimality criterion. The parameters as needed in this thesis are summarized in Table 1.1. E.g., $Pm|\mathsf{p}_j = 1, \mathsf{r}_j|C_{\max}$ denotes a machine scheduling problem with m parallel machines where each job has unit processing time and a release date while the objective is to minimize the makespan. As another example, $P|\mathsf{size}_j|\sum_j w_j C_j$ denotes a parallel machine scheduling problem where each job j requires size_j machines during its processing interval and the objective is to minimize the sum of weighted completion times.

	n	number of jobs
	m	number of machines
α	P	parallel machines, here m is not part of the input
	Pm	m parallel machines, m is part of the input
	p_j	processing time of job j
	r_j	release date of job j
	d_j	deadline of job j
β	size_j	number of machines required by job j
	prec	precedence constraints
	pmtn	preemption is allowed
γ	$C_{\max}$	makespan minimization
	$\sum w_j C_j$	weighted completion time criterion

Table 1.1: Notation for machine scheduling problems.

1.2.2 Project scheduling

Project scheduling problems have several additional characteristics compared to machine scheduling problems. Each job may be processed in different ways (multiple modes), or the execution of a job may need several resource units of different resources (renewable and non-renewable). The $\alpha|\beta|\gamma$-classification scheme has therefore been extended by Brucker et al. [36]. In this scheme, RCPSP can be written as $MPS\rho, \infty|prec|C_{\max}$ for multi-project scheduling of renewable resources with capacity ρ, precedence constraints and the objective to minimize the makespan. In Tables 1.2 and 1.3 the notation for project scheduling problems as used throughout this thesis is summarized.

	n	number of jobs
α	ρ	number of resource constraints
	$\mathcal{J}$	set of jobs
	MPS	multi-mode project scheduling
	p_j	processing time of job $j \in \mathcal{J}$
	r_{jk}	resource demand (consumption) of job j of resource k
	r_j	resource demand of job j if resource is clear
	p_{jm}	processing time of job j in mode m
β	r_{jmk}	resource demand of job j in mode m of resource k
	R_k, R	resource capacity of resource k, k omitted if resource is clear
	$\prec$	precedence constraints
γ	$C_{\max}$	makespan minimization
	$\sum_k \max r_k(S,t)$	resource availability cost

Table 1.2: Notation for project scheduling problems in extended $\alpha|\beta|\gamma$-scheme.

$\text{est}_j, \text{lst}_j$	earliest and latest start of job j
$\text{ect}_j, \text{lct}_j$	earliest and latest completion time of job j
e_j	energy of job j; $e_j = r_j \cdot p_j$
$G = (V, E)$	precedence graph with node set V and directed edge set E
$\mathcal{M}_j$	set of modes of job j

Table 1.3: Further notation used for project scheduling problems when considering CP and IP techniques.

1.3 Solving techniques

Many scheduling problems are well known for their hardness to be solved to optimality. Hence, they have motivated researchers to develop different approaches in order to tackle these problems. The most prominent approaches are integer programming (IP), constraint programming (CP) and satisfiability testing (SAT). We will introduce these in the following together with a hybrid approach called *Constraint Integer Programming (CIP)*.

1.3.1 Mixed Integer Programming

In Mixed Integer Programming (MIP) we are given $n_x \in \mathbb{N}$ integer variables $x \in \mathbb{Z}^{n_x}$ and $n_y \in \mathbb{N}$ continuous (for computational issues rational) variables $y \in \mathbb{Q}^{n_y}$, an inequality system $A[x,y]^t \leq b$ with $A \in \mathbb{Q}^{m \times (n_x+n_y)}$ and $b \in \mathbb{Q}^m$ with $m \in \mathbb{N}$ inequalities.

Definition 1.1. *The polyhedron* $P(x,y) := \left\{(x,y) \mid A[x,y]^t \leq b, x \in \mathbb{Z}^{n_x}, y \in \mathbb{Q}^{n_y}\right\}$ *is called the* set of feasible solutions*.*

By introducing cost coefficients $[c_x, c_y] \in \mathbb{Q}^{n_x+n_y}$, linear objective functions can be modeled that are to be minimized or maximized. We mainly consider minimization problems here and therefore restrict the definitions to that case.

Definition 1.2. *A* Mixed Integer Program (MIP) *in minimization form is given by the following formulation:*

$$\min \left\{c_x x + c_y y \mid (x,y) \in P(x,y)\right\}. \tag{1.6}$$

In case that $n_y = 0$, we speak of an Integer Program (IP), and in case that $n_y = 0$ and $x \in \{0,1\}^{n_x}$ of a Binary Program (BP).

Solving BPs is strongly $\mathcal{NP}$-hard as shown by Karp in 1972 via reduction from 3-SAT [155]. Moreover, Zuckerman [272] showed in 1996 that a constrained version is inapproximable within any constant factor unless $\mathcal{P} = \mathcal{NP}$.

Definition 1.3. *The* Linear Programming (LP) Relaxation *of a MIP is given by relaxing the integrality constraints on the integer variables:*

$$\min \left\{c_x x + c_y y \mid A[x,y]^t = b, x \in \mathbb{Q}^{n_x}, y \in \mathbb{Q}^{n_y}\right\}.$$

This relaxation provides dual bounds (in case of minimization lower bounds) on the objective value of the MIP. It can be solved in polynomial time via the Ellipsoid method [130, 157]. In practice, it is nevertheless solved via the (in general non-polynomial) Simplex Method [69] due to its experimentally faster convergency.

1.3.1.1 Solving a MIP

General purpose MIP solvers are IBM ILOG CPLEX [68] and Gurobi [199] that use a *branch-and-bound* framework where a rooted tree is built up step by step. In each node of the tree (starting from the root) the LP relaxation is solved. If the dual bound at least matches the best primal bound, the node can be pruned (*bounding*). If all values $x^\star$ of the LP solution $(x^\star, y^\star)$ are integral, a primal solution has been found. Otherwise, fractional solution values from a solution to the LP relaxation can be used to split the feasibility space into two disjoint subspaces (*branching*), while the union of both still contains all feasible solutions of the current node. Branching is performed by selecting a variable x_j with fractional value $x_j^\star$ and two children of the current node are created. They are called the *left* and the *right* child. The left (right) child node obtains the additional constraint $x_j \leq \lfloor x_j^\star \rfloor$ ($x_j \geq \lceil x_j^\star \rceil$). After branching is performed, an unpruned node is selected and the same process that begins by solving the LP relaxation is started over again.

Valid inequalities (*cutting planes*) can be added in order to strengthen the LP relaxation. Such inequalities cut off non-integer points, e.g., the current fractional LP solution.

The most prominent and basics cutting planes are *Gomory-Chvàtal cuts* [61, 125]. For an overview on cutting planes and their relations we refer to Cornuéjols [66] and Fischetti and Lodi [111].

Besides cutting plane algorithms, further techniques have been developed to solve Integer Programs. Among them are Lagrangean relaxation, Benders decomposition [28] and Dantzig-Wolfe decomposition [70, 71, 88]. We refer to Section 5 for an application of Dantzig-Wolfe decomposition to a multi-mode resource leveling problem.

1.3.1.2 IP formulation for RCPSP

Several exact IP formulations for RCPSP exist (time-indexed, event- and flow-based) [164]. In our study, we use one of the first formulations by Pritsker et al. [209]. The reason for choosing this formulation is due to the strong dual bounds, the various improvements suggested from literature and the close relation of the resource constraints in that formulation to knapsack constraints.

The formulation by Pritsker et al. is based on binary variables x_{jt} that are one if job j starts at time t and zero otherwise. This formulation is only pseudo-polynomial due to the time discretization but for low makespans the obtained lower bounds are remarkably good.

$$\min \quad \sum_t t \cdot x_{n+1,t} \tag{1.7}$$

$$\text{subject to} \quad \sum_t t \cdot x_{it} + p_i \leq \sum_t t \cdot x_{jt} \qquad \forall\,(i,j) \in E \tag{1.8}$$

$$\sum_t x_{jt} = 1 \qquad \forall\, j \in \mathcal{J} \tag{1.9}$$

$$\sum_{j \in \mathcal{J}} \sum_{\tau = t - p_j + 1, \tau \geq 0}^{t} r_{jk} x_{j\tau} \leq R_k \qquad \forall\, k \in \mathcal{R}, \forall\, t \tag{1.10}$$

$$x_{jt} \in \{0, 1\} \qquad \forall\, j \in \mathcal{J}, \forall\, t \tag{1.11}$$

In the objective function (1.7), the start time of the dummy job $n+1$ for the project end is minimized. Constraints (1.9) and (1.11) ensure that each job is scheduled exactly once, while constraints (1.8) enforce the precedence relations and constraints (1.10) model the resource constraints.

1.3.2 Constraint Programming

Constraint Programming (CP) is a programming paradigm wherein a problem is modeled via variables and logical constraints. This model is solved by using the logical implications between the variables that are induced by the constraints in order to tighten the variable domains throughout an enumerative branch-and-prune search.

In CP, a problem is formulated as a *Constraint Satisfaction Problem* (CSP). A CSP consists of a set of variables $\{x_1, \dots, x_n\}$ with a domain per variable and a set of constraints.

Definition 1.4 (Domain)**.** *A domain $D(x_i)$ is the set of all values which variable x_i is allowed to take. If the domain is an interval, we write $D(x_i) = [\ell\mathrm{b}_i, \mathrm{ub}_i]$ with $\ell\mathrm{b}_i < \mathrm{ub}_i$ and $\ell\mathrm{b}_i, \mathrm{ub}_i \in \mathbb{Q}$.*

In case that the lower and upper bound are equal, we say the variable is *fixed*.

Definition 1.5 (Constraint). *A constraint $C(X)$ is an n-ary relation over a sequence of variables $X = x_1, x_2, , \ldots, x_n$ and their respective domains $D(x_1), D(x_2), ...D(x_n)$. Hence, a constraint describes the solution space by a subset of the Cartesian product over the domains $C(X) \subseteq D(x_1) \times D(x_2) \times \ldots, \times D(x_n)$.*

Observe that a constraint gets as input a set of variables together with their respective domains and describes the set of all solutions (all variable assignments) that satisfy this constraint. Given a variable assignment, i.e., all variables are fixed, we say $C(X)$ *holds* to describe whether under this assignment the solution space of this constraint is not empty.

Example 1.2. Basic examples for constraints are:

- Linear constraints: $C_1(\{x_1, x_2\}) = \{x_1, x_2 \mid x_1 + 2x_2 \leq 8\}$
- Quadratic constraints: $C_2(\{x_1, x_2, x_3\}) = \{x_1, x_2, x_3 \mid x_1 x_2 + x_3 \leq 11\}$
- Set-partitioning constraints:
  $C_3(\{x_1, x_2, x_3\}) = \{x_1, x_2, x_3 \in \{0, 1\} \mid x_1 + x_2 + x_3 \leq 1\}$
- Cumulative constraints:

$$C_4(\boldsymbol{S}) = \texttt{cumulative}(\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}, R) = \left\{ \boldsymbol{S} \in \mathbb{N}_0^n \mid \sum_{j:S_j \leq t < S_j + p_j} r_j \leq R, \forall\, t \right\}.$$

Definition 1.6 (Constraint Satisfaction Problem (CSP)). *A* Constraint Satisfaction Problem*, CSP=$(\mathcal{C}, X, D)$, consists of a set of variables $X = \{x_1, x_2, \ldots, x_n\}$ with their respective domains $D(x_1), D(x_2), \ldots, D(x_n)$, and a set of constraints $\mathcal{C}$ in which each constraint $C \in \mathcal{C}$ is defined over a subsequence of variables $X_C \subseteq X$.*

We define the feasible region of a CSP as:

$$CSP = \left\{ X = \{x_1, \ldots, x_n\} \mid x_i \in D(x_i), \forall\, i, X \subseteq \bigcap_{C \in \mathcal{C}} C(X_C) \right\}.$$

CSPs are strongly $\mathcal{NP}$-hard in general, because they generalize the SAT problem [64]. Typical examples for CSPs are the eight queens puzzle, crosswords, sudoku and the knapsack problem.

1.3.2.1 Solving a CSP

Prominent CP solvers are CHiP [90], Choco [58], Comet [63], Gecode [119] and IBM ILOG CP Optimizer [198].

A CSP is solved similarly to a MIP via a search tree using propagation algorithms instead of LP relaxation. The main difference to MIP is that in CP no objective function is at hand. Proving feasibility or infeasibility is the goal in each node (called *candidate*) of the search tree. An objective function can be modeled via a binary search on an artificial variable together with a linear constraint. Furthermore, no LP relaxation, no fractional variables and no dual bounds implicitly exist. It is important for any CP solver

to efficiently check a partial assignment for global feasibility. This checking is usually done constraint-wise.

The main workhorse in CP is *domain propagation*, a technique that reduces the domains of the variables or generates new constraints that can be propagated themselves. If the domain of a variable gets empty, the candidate is *abandoned* (the node is pruned) and we backtrack to the last feasible node. If no further domain reductions can be found and none of the domains is empty, a branching is performed. As a branching candidate, usually a variable is picked that is contained in many constraints and has been involved before in successful propagations. Instead of branching on one variable, sets of variables (e.g., set-partitioning constraints) can be used as branching decision, or new constraints can be added that split the search space. After branching, the next candidate (node) to be considered is selected which is often based on an estimated number of domain reductions. In CP, a depth-first-search is used.

1.3.2.2 Domain propagation

In each node of the search tree, domain propagation is performed. Given a partial assignment of the variables, the goal is to remove inconsistent values from the domains of the variables. This is done by analyzing repeatedly the variable domains and their interdependencies that are induced by the constraints. If only one constraint is involved in the propagation, this process is also called *constraint propagation*.

By removing inconsistent values, a certain type of consistency can be achieved. Different concepts of consistency exist, see e.g. [166], and propagation algorithms are clustered according to the *consistency* they achieve. In a k-consistency test, constraints with k variables are considered, also known as node- ($k = 1$), arc- ($k = 2$) and path- ($k > 2$) consistency [113, 189, 249]. Domain consistency tests check each value from the domain of a variable whether it can be excluded, if no feasible assignment of all other variables exists anymore [65]. For bound-consistency, only the left and right bound of a domain is considered [73, 175, 190, 253]. Bound-consistency can sometimes be achieved in polynomial time. E.g., the propagation algorithms from Section 2.1.3.2 for the `cumulative` constraint achieve bound consistency.

As a consequence, for $\mathcal{NP}$-hard problems not all inconsistent assignments can be efficiently removed from the variables domains. Hence, fast heuristics or approximations are needed. Algorithms that remove these inconsistent values are called *propagation algorithms*, in some former papers referred to as *consistency tests*. To summarize, propagation algorithms operate on a single constraint or on a set of constraints, they check a certain condition and if this is satisfied, the domain of one or more variables can be reduced.

We will formally introduce the terms propagation algorithms and domain reductions and also review the main propagation algorithms in cumulative scheduling in Sections 2.1.1 and 2.1.3.2.

1.3.2.3 CP formulation for RCPSP

RCPSP can be modeled as a constraint program using the `cumulative` constraint [5], which enforces resource-feasibility for a resource k. We denote by $\boldsymbol{S} = [S_j]_j$, $\boldsymbol{p} = [p_j]_j$

and $\boldsymbol{r}_k = [r_{jk}]_j$, the vectors of start times, processing times and resource demands.

$$\texttt{cumulative}(\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}_k, R_k) : \left\{ \boldsymbol{S} \in \mathbb{Z}_0^n \mid \sum_{j: S_j \leq t < S_j + p_j} r_{jk} \leq R_k, \, \forall \, t \right\}.$$

Precedence constraints $(i, j) \in E$ are modeled via the `precedence` constraint (in SCIP called `varbound`):

$$\texttt{precedence}(S_i, S_j, p_i) : \{(S_i, S_j) \mid S_i + p_i \leq S_j\}.$$

Then, RCPSP can be modeled as a CP as follows:

$$\begin{array}{lll} \min & C_{\max} & \\ \text{subject to} & \texttt{precedence}(S_i, S_j, p_i) & \forall \, (i, j) \in E \\ & \texttt{cumulative}(\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}_k, R_k) & \forall \, k \in \mathcal{R} \\ & D(S_j) = \mathbb{N}_0 & \forall \, j \in \mathcal{J}. \end{array}$$

We point out that no additional variables need to be introduced. The constraints capture the logic given by the resource constraints.

1.3.3 Satisfiability testing (SAT)

In an instance of SAT we are given a set of boolean variables $\{x_1, \ldots, x_n\}$ (that can take the values *true* and *false*) and boolean operators AND ($\wedge$), OR ($\vee$) and NOT ($\neg$) that connect these variables with each other to form boolean expressions. Let $\{x_1, \ldots, x_n\}$ be the set of boolean variables, where we identify the *false* assignment with 0, and a *true* assignment with 1. A *literal* y_i is a variable x_i or its negation $\neg x_i$. E.g., x_1 and $\neg x_1$ are two literals.

Definition 1.7. *A* clause *is a disjunction of literals.*

E.g., $(\neg x_1 \vee x_2)$ is a clause. Given this, we can define the SAT problem formally.

Problem: **Satisfiability (SAT)**
Instance: A set of m clauses over boolean variables $x_1, \ldots, x_n$.
Question: Is there an assignment $x^\star \in \{0, 1\}^n$ that satisfies all clauses?

E.g., $(\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_3) \wedge (x_2 \vee \neg x_3)$ is an instance of SAT in *conjunctive normal form.*

By the famous Cook-Levin theorem, SAT is known to be the first $\mathcal{NP}$-complete problem [64]. If all clauses contain exactly three variables, the problem is called 3SAT and is already strongly $\mathcal{NP}$-complete. Many decision problems can be easily expressed as an instance of SAT. Hence, given some efficient algorithm to solve SAT, these problems can be solved generically. This has guided the rapid evolution of SAT solvers.

1.3.3.1 Solving an instance of SAT

Examples for modern SAT solvers are CHAFF [191], GRASP [180], and MINISAT [103, 104]. They rely on a branching scheme, like the DPLL-algorithm [74, 75] from 1960/62 that enumerates all possible variable assignments. Here, *unit domain propagation* [268]

is applied in every node, like constraint propagation per clause. Infeasible subproblems are analyzed in order to learn new clauses and to perform *non-chronological backtracking* [120]. Furthermore, *restarts* are used in order to explore promising search spaces more frequently and to perform rather expensive propagations only in the root node.

The search process in SAT works as follows: Initially all variables are unassigned. In each node of the search tree, *unit domain propagation* is performed, i.e., constraint propagation over each boolean formula is applied. Thereby, variables of one constraint (one formula) are checked whether they can be fixed to 1 (true) or 0 (false). E.g., if all but one literal of a clause are false, the remaining variable can be set such that the clause is satisfied. Local search heuristics are used in order to find a feasible assignment. As soon as the first feasible assignment has been found, the algorithm stops. When constraint propagation cannot detect any further fixings, some variable is fixed to true or false (the branching step). The branching variable is chosen according to its importance in former propagations and its involvement in conflicts. If all literals of at least one clause are fixed to zero, the subproblem is infeasible. This is called a *conflict*. Conflicts are analyzed in order to produce *conflict clauses* [180]. During this analysis, a conflict graph is created. Such a graph is a logical expression tree where vertices correspond to variable fixings (zero or one). There is a directed edge in that graph between two vertices u and v if the fixing of the variable belonging to u deduced the fixing of the other variable in node v. Cuts in that graph that separate the branching decisions from the infeasibility yield new clauses. Furthermore, the analysis may show that some branching decision (variable fixing) in the last subtree at depth level ℓ yielded the infeasibility together with other fixings at some depth level $\ell' < \ell$. Then, the negation of the branching decision can be already applied at depth level ℓ'. This is called *non-chronological backtracking*. The concept of conflict analysis in our framework with integer variables is explained in more detail in Section 2.2.1.

We remark that in contrast to IP and CP solver, a SAT solver creates exactly one child per node. After applying conflict analysis and backtracking a second child is created if needed. Hence, the search tree as kept in memory is always a path.

We do not state a SAT model for RCPSP since it will not be used in this thesis and similar to the IP models different formulations exist. We refer to Horbach [144] for further readings.

1.3.4 Constraint Integer Programming

All three paradigms presented so far share some characteristics like *branching* and *pruning*. While in MIP, branching decisions are based on the values of the variables in the LP relaxation and the objective function, in CP and SAT this decision is made based on the information gathered throughout search: involvement in propagation (CP, SAT) and in conflict clauses (SAT).

Learning from the other techniques and hybridization already have a long history. In particular, conflict analysis from SAT has been generalized to integer variables for CP and to LP solving [2]. To fully hybridize the three techniques means to use a unifying branching strategy, to perform domain propagation, solving an LP relaxation, and to apply conflict analysis and restarts. One of the closest integrations is the *Constraint Integer Programming (CIP)* framework SCIP by Achterberg [2] that we use for our studies. Another hybrid framework is given by the *branch-infer-and-relax* framework SIMPL [11].

Formally, we define a *Constraint Integer Program* as follows.

Definition 1.8. *A* Constraint Integer Program*, CIP=$(\mathcal{C}, X, D, c)$ consists of a set of variables $X = \{x_1, x_2, \ldots, x_n\}$ with their respective domains $D(x_1), D(x_2), \ldots, D(x_n)$, and a set of constraints $\mathcal{C}$ in which each constraint $C \in \mathcal{C}$ is defined over a subsequence of variables $X_C \subseteq X$ with objective function vector $c \in \mathbb{Q}^n$.*

$$\min \left\{ \sum_{i=1}^{n} c_i x_i \mid x_i \in D(x_i), \forall\, i \wedge X \subseteq \bigcap_{C \in \mathcal{C}} C(X_C) \right\}. \tag{1.12}$$

Furthermore, we require for a CIP that after fixing all integer variables, the remaining problem is an LP.

SCIP integrates the three techniques in one search tree, i.e., in every node of the tree an LP relaxation is solved and domain propagation is performed. Furthermore, primal heuristics and more problem specific propagation algorithms can be applied. If a node is detected to be infeasible, conflict analysis is applied in order to generate conflict clauses and perform non-chronological backtracking. Conflict analysis can also be applied to infeasible LPs. We will use this framework in our experiments.

It is best to understand the CIP model, by thinking of a Constraint Satisfaction Problem that is handed to the solver. Each constraint (precedence or cumulative in case of scheduling) has the ability to perform propagation and separation algorithms. It is essential to understand that not all inequalities must be added to the LP relaxation. Even any kind of problem specific relaxation can be used. In particular, it must not be based on an exact IP formulation. Therefore, the constraints must be able to efficiently check feasibility of any presented solution. Hence, it is important to verify efficiently (in low order of polynomial time) whether a solution violates a constraint or not. The faster this can be done, the better for the overall running time. Summarizing, the constraints are the main part of a CIP solver.

Hybrid branching schemes [3] can be used in this framework: the scores from IP (pseudo-cost score), CP (inference score, the number of involved domain reductions) and SAT (conflict score, number of conflicts involved) are weighted into one score and the best weighted candidate is selected, see Section 3.3 for further readings. E.g., in SCIP a huge weight is placed on the conflict score and the inference score serves as tie-breaker.

When considering integer variables in SAT solvers, not only the variable but also the value to branch on is chosen by the conflict score. In a SAT model each possible value of an integer variable is encoded as a halfspace: $\{S_j \leq 4\}, \{S_j \leq 5\}, \ldots$. Conflict scores per variable and lower and upper bound values are called VSIDS (variable state independent decaying sum) and are increased whenever a variable bound is reported to the conflict graph [191].

CIP formulation for RCPSP The CIP formulation we use reads as the CP formulation. The difference is that the constraints additionally capture the IP logic and are able to add variables and separate cutting planes and thereby provide an LP relaxation. E.g., the cumulative constraint may add binary variables and all necessary constraints to the model in order to use the LP relaxation of the exact IP formulation by Pritsker

et al. [209]. For completeness, the CIP model reads as:

$$\begin{array}{rll} \min & C_{\max} & \\ \text{subject to} & \texttt{precedence}(S_i, S_j, p_i) & \forall\, (i,j) \in E \\ & \texttt{cumulative}(S, p, r_k, R_k) & \forall\, k \in \mathcal{R} \\ & D(S_j) = \mathbb{N}_0 & \forall\, j \in \mathcal{J}. \end{array}$$

1.4 Related work

Approaches to solve RCPSP range from inexact approaches, such as ordering heuristics or genetic algorithms, to exact approaches, such as Mixed-Integer Programming, Satisfiability Solving and Constraint Programming. Next, we give a short overview on the successful implementations which will be refined later in the corresponding sections.

Main techniques

Heuristics On the heuristic side, excessive research has been carried out such that numerous algorithms have been developed. Surveys can be found by Hartmann and Kolisch [38, 135, 136, 160, 161]. Among these heuristics are *schedule generation schemes* (SGS) that work similar to a list scheduling algorithm for machine scheduling problems. We consider two schemes, the *serial SGS* and the *parallel SGS*, also see [160] for more details. In a serial SGS, jobs are considered in a topological order (e.g., sorted by their earliest start or by the minimum float time) and are scheduled according to that order as early as possible, respecting the precedence and resource constraints. In a parallel SGS, the resource profiles are focussed on. For each resource profile (one per constraint) the first point in time is considered when enough capacity is available to schedule a job. Then, a job is selected that can be scheduled there with respect to the precedence constraints.

The serial generation scheme has been extended to a *forward-backward* SGS by Li and Willis [176], also referred to as *bidirectional* SGS. The first schedule, the so-called *forward schedule*, is obtained by performing a serial SGS. Then, all jobs are sorted in non-increasing order of their completion times in the forward schedule. According to that ordering, a backward schedule is created by scheduling all jobs as late as possible. It gets clear from the way the algorithm works, that if a feasible forward schedule has been found, a feasible backward schedule can be obtained, since no job needs to be scheduled earlier as in the forward schedule. Recreating a forward schedule by sorting the jobs according to their start times in the backward schedule leads to a makespan not larger than the one in the forward schedule. We use such a bidirectional SGS in our computations.

Genetic algorithms use these generation schemes as a subroutine to generate a schedule for a given population (an order among the jobs that respects the precedence constraints), see e.g. [133, 264]. Furthermore, tabu search approaches [195, 202, 247] have been developed in order to avoid evaluating an order multiple times. In tabu search in contrast to local search, when escaping from a local optimum, some neighboring solutions are declared as *tabu* to avoid cycling. In order to not explore similar solutions, a small tabu list is kept in memory.

More elaborate heuristics have also been investigated. A large neighborhood search has been employed by Palpant et al. [201]. They fix a certain amount of the variables or

heuristically add precedence constraints to the model and solve the remaining problem (still an RCPSP) via CP or IP techniques.

MIP Several IP formulations have been proposed for RCPSP. The classical formulation goes back to Pritskers et al. [209]. For an overview and an experimental study on other types of formulations such as time-indexed, event- and flow-based formulations we refer to Koné et al. [164].

We briefly mention the most important IP techniques besides the formulation presented by Pritsker et al. [209]. Cutting plane algorithms have been used by Christofides et al. [59] and Sankaran et al. [217] on the standard time-indexed formulation. E.g. in [59], it remains unclear whether using disaggregated precedence constraints yields a total speed up of the basic formulation. As reported by Uetz [250] this seems to be instance dependent. On the other hand, new valid inequalities for the LP relaxation based on constraint based arguments are derived in [81] such as shaving and clique cuts which yield stronger dual bounds.

Order based formulations have been proposed that introduce binary decision variables whether one job is executed after another. In case of cumulative scheduling problems, jobs may be allowed to run in parallel. Hence, to exclude only those that violate the capacity Bartusch et al. [27] interpret RCPSP as the search for a strict order between subsets of the jobs which correspond to feasible left-shifted schedules.

This idea has been further extended by Alvarez-Valdés and Tamarit [6]. A set of jobs that exceeds the capacity, if scheduled pairwise in parallel, is called a *forbidden set*. Such a set is minimal if deleting any of the jobs from the set would not violate the capacity. As there may be exponentially many of these sets and several are not of interest as precedence constraints may be present between two jobs, Stork and Uetz [246] use a tree structure to enumerate all necessary minimal forbidden sets. This idea has been pursued further. Interpreting the strict order from the transitive closure of the precedence network, a schedule corresponds to a resource flow [13], where the available resource capacity *flows* through the precedence network and a maximum cut in that network needs to be always less than the capacity in order to yield a feasible schedule. One drawback of these two models is that big-M constraints are used to model the precedence relations. Furthermore, the resource flow model induces symmetry and yields a weaker LP relaxation. This model has been used successfully to measure the robustness of a schedule with respect to uncertain processing times, see Leus and Herroelen [174].

Mingozzi et al. [186] propose a preemptive relaxation based on feasible subsets. This approach has been improved by Brucker and Knust [37] by using column generation and allowing preemption. Similarly using feasible subsets, Carlier and Néron [42] compute linear lower bounds of a linear multi-elastic preemptive relaxation.

For the case of generalized precedence constraints, Bianco [31, 32] derives lower bounds by relaxing resource constraints for jobs which are not precedence related. More precisely, they model resource conflicts between two jobs via edges in an expanded activity-on-node network. This allows a dynamic programming approach to compute valid lower bounds. Lower bounds derived from a Lagrangean Relaxation also prove useful, see Möhring et al. [188]. The authors show that the Lagrangean subproblem can be efficiently solved via a sequence of maximum flow computations. Furthermore, they use the (possibly infeasible) start times from the relaxation as a starting point for a heuristic based on α-points, see [239]. Later on, similar approaches using Lagrangean

relaxation are used to derive lower bounds or solve pricing problems of other formulations for RCPSP, see e.g. [37, 80].

As time-indexed formulations are only of pseudo-polynomial size, they become intractable if the makespan gets too large. Two event-based formulations, called *start/end event-based* and *on/off event-based*, have been proposed by Koné et al. [164] to overcome this disadvantage. But the dual bounds of this relaxation are weak. We elaborate on these formulations in Section 1.5.

CP Constraint Programming techniques use logical implications to strengthen the bounds of variables via domain propagation. These techniques are also used as a preprocessing routine in IP solving in order to reduce the number of variables. In case of scheduling, a vast amount of literature can be found on propagation algorithms. We will recapitulate the most important ones in Section 2.1.3.2.

The most prominent solving approaches for RCPSP based on propagation have been proposed by Klein and Scholl [159], Caseau and Laburthe [47], Baptiste et al. [21] and Dorndorf et al. [95]. More recent works can be found in Liess and Michelon [177], Vilím [263] or Artigues et al. [12]. In [12], several approaches are compared with each other.

A pure CP approach is presented by Lies and Michelon [177] who use lazy constraint generation (like cutting planes in IP). Throughout search, when a subset of the jobs violates the capacity, a resource constraint involving only this subset is created. This technique substitutes the time-tabling propagation algorithm as it posts new constraints exactly when time-tabling would have detected an infeasibility.

Worth-mentioning on the CP side is the work of Laborie [168], where minimal critical sets (MCS) are resolved during branching by adding local precedence constraints. He uses time-tabling, edge-finding and shaving as propagation rules. At that time, more than 31% of the best known lower bounds on a public library (PSPLib) have been improved and more than 15% of open problems have been closed.

Applications of CP techniques to scheduling problems can be found e.g. in Dorndorf et al. [94] in a job-shop scheduling environment where they show that guessing the direction of the edges of the disjunctive graph leads to better heuristic solutions within the same amount of computation time than other approaches. Approaches that combine heuristics with CP techniques have been reported by Nuijten and Aarts [197], Pesch and Tetzlaff [205], Phan Huy [92], and by Nuijten and Le Pape [196].

SAT Satisfiability testing, or SAT for short, also heavily relies on domain propagation algorithms embedded in a search process, and additionally learns from infeasible states by analyzing the infeasibility to use *non-chronological backtracking* or to create new local or global valid clauses. By these new clauses, similar search states can be eliminated early. The underlying model is an instance of SAT and therefore RCPSP needs to be reformulated into SAT clauses. Horbach [144] proved this to be a very efficient and successful approach for RCPSP. Furthermore, Ansótegui et al. [8] show how time-indexed, task- and flow-based formulations can be encoded in a SAT solver. Their approach is based on satisfiability modulo theories (SMT) which generalizes SAT solving by adding arithmetic and other theories. They present competitive results which show that time-indexed formulations are well suited for small instances.

Hybrids Several of the approaches above already combine CP and IP techniques by using constraint propagation to obtain tighter IP relaxations. Such hybrids of IP and CP can be found by Baptiste and Demassey [19], Brucker et al. [39] and Demassey et al. [81]. In particular, new valid inequalities for the LP relaxation based on constraint based arguments are derived in [81] such as shaving and clique cuts.

Hybrid solvers that combine the strengths from the different communities, like a CP-SAT hybrid [228] or the CIP framework SCIP [137], prove most successful in order to solve hard scheduling instances efficiently. Schutt et al. [228] use a CP approach with time-tabling, edge-finding and not-first not-last detection [230] as the main propagation algorithms. They generate a SAT model that discovers conflict clauses throughout search. Currently, their implementation is the fastest on PSPLib, though there are few instances where the SAT model by Horbach [144] or our approach provide better primal and dual bounds.

Our results have been developed independently in parallel to these works.

1.5 Benchmark instances and computational studies

In this section we describe the publicly available instances and some indicators by which these instances are characterized. In Section 1.5.2, we collect known results from the literature in order to give the reader a familiarity with the approaches that have already been applied and empirically compared. We conclude by describing the computational environment used in the next two chapters for tuning our CIP framework for RCPSP.

1.5.1 Benchmark instances and instance indicators

One of the first official benchmark sets is given by the *Patterson* set [203]. Unfortunately, this set turned out to be relatively easy to solve. Several indicators such as precedence ratio or resource strength have been developed in order to cluster scheduling problems, to estimate their hardness and to apply appropriate techniques depending on the given instance. Nevertheless, none is perfectly suited to select the best propagation algorithm throughout a branch-and-bound search. In the following, we mention the most important indicators.

The *precedence ratio* (also called *order strength* [182], *flexibility ratio*[78] or *density* [78]) is given by the number of pairs of jobs that are transitively precedence related divided by the total number of pairs. A precedence ratio of zero means that all jobs can be executed in parallel (according to the precedence constraints), while a ratio of one means that all jobs are totally ordered. With increasing precedence ratio the hardness decreases as more scheduling decisions are already fixed.

The *resource strength* (per resource constraint), see e.g. Kolisch et al. [163], combines precedence relations with resource information. It is defined as $RS = (R_k - R_k^{\min})/(R_k^{\max} - R_k^{\min})$. Here, $R_k^{\min} = \max\{r_{jk} \mid j \in \mathcal{J}\}$ is the maximum demand of any job of that resource constraint and $R_k^{\max}$ is the maximum peak in an earliest start schedule when neglecting resource capacities. Hence, the resource strength can be seen as a normalized resource capacity, i.e., the strength is close to one if an earliest start schedule satisfies the resource constraint and close to zero if either many small jobs can be executed in parallel or at least one job needs all the capacity. Easy instances have resource strength zero or one, while those with resource strength zero are much harder.

Hard instances have an intermediate resource strength. Altogether a bell-shaped easy-hard-easy curve can be observed. Artigues et al. [12] present further indicators and show in an experimental study that none fits best in order to describe the hardness of an instance or to decide prior to search which algorithm to use.

Nevertheless, the *disjunction ratio* [21] works well as indicator to distinguish instances. This indicator combines precedence and resource features and is computed as follows: $DR = |\bar{E} \cup D|/(n \cdot (n+1)/2)$, where $\bar{E}$ denotes the transitive closure of the precedence graph and D is the set of pairs of jobs that are in disjunction, i.e., $D := \{(i,j) \in \mathcal{J}^2 \mid \exists\, k \in \mathcal{R} : r_{ik} + r_{jk} > R_k\}$. If the disjunction ratio is high, we speak of *disjunctive* instances. Therein, pairs of jobs that cannot be scheduled in parallel occur frequently. Strong lower bounds on the makespan can be computed by considering the sum of processing times of all cliques formed by the disjunctive jobs. For example, the Patterson set turned out to be disjunctive with high precedence ratio and is therefore easy to solve. *Cumulative* instances have a low disjunction ratio and are characterized by the fact that many jobs can be scheduled in parallel, such that almost a two-dimensional packing problem needs to be solved if there were no precedence constraints at hand. Here, volume arguments (like in propagation algorithms such as energetic reasoning) and the LP relaxation provide good lower bounds on the makespan, but it is harder to find optimal solutions by heuristics due to the packing character.

Besides the Patterson set, further instances have been generated with respect to indicators such as resource strength and precedence ratio. The nowadays well-established library for RCPSP, the PSPLib [162], has been created by Kolisch et al. [163]. Four sets (J30, J60, J90 and J120) with 30, 60, 90 and 120 jobs per instance are given with 2040 instances in total. Each instance of a set contains 4 resources and the jobs demand between 1 and 10 resource units of multiple resources. Some instances have a high resource strength and those that are unsolved yet, are rather disjunctive than cumulative.

Another set, the Pack instances [43], are of highly cumulative type, i.e., there are only few precedence constraints and in a feasible schedule several jobs are scheduled in parallel. The set contains 55 instances, the number of jobs, on average 25, varies from 17 to 35 and the number of resource constraints is three, while the capacity of each resource ranges from 5 to 10. The instances are of two types. The first type may include disjunctions between the jobs as resource demands have been generated randomly between zero and the capacity. In the second type of instances, all demands cannot exceed half the capacity, hence no disjunctions are present. That's why, these instances are called highly cumulative. They are hardest to solve as they have low order strength and zero disjunction ratio. Due to the low makespan and only up to 35 jobs, IP models derive good bounds on these instances. Only 79% of the Pack instances have been solved to optimality. We will close some of these instances for the first time.

Though extensive research has been carried out on RCPSP and all the indicators mentioned above have been used to characterize the hardness of an instance, it has been computationally revealed that the initial duality gap is the most discriminating indicator. E.g., Artigues et al. [12] show that a standard MIP has an initial duality gap of 27% on average on set J60 while unsolved instances bear an integrality gap of 62% on average. Similarly, the Pack instances that are of highly cumulative type bear a high integrality gap. For the other indicators the following observations have been made: Hardness decreases with increasing order strength and network complexity, and decreasing resource factor. Highly disjunctive problems and highly cumulative problems are hard.

1.5.2 Computational studies

In many of the aforementioned works different approaches and formulations have been compared with each other. We collect the most important insights here. E.g., Koné et al. [164] compare the standard MIP by Pritsker et al. [209] using the strong precedence inequalities from [59], denoted by (DT), with the continuous flow formulation (FCT) [13], the start/end event-based formulation (SEE), and with the on/off event-based formulation (OOE) [164]. Table 1.4 indicates the strength of the time-indexed MIP for small instances and the strength of the on/off event-based MIP for large scale instances. In the scaled instances, the processing times from a subset of the jobs has been scaled by a factor of 50.

Table 1.4: Comparison of the number of optimally solved instances by time-indexed and event-based approaches from [164].

	DT	FCT	SEE	OOE
J30	82%	62%	2.9%	30%
Pack	75%	0%	0%	9%
Pack_{50}	0%	7%	4%	18%

The fact that time-indexed MIPs are no longer capable to solve instances with high makespan, motivates us to invest a theoretical and computational study on a different kind of relaxation. We present a *continuous relaxation* for cumulative scheduling problems in Section 2.4 which is able to handle even large real-world instances as shown in Section 4.1.

Now we turn to the different solution methodologies. Several exact approaches, which mainly rely on intelligent enumeration, exist and have been developed by the IP, CP and SAT communities. It is widely believed that RCPSP is best solved via CP and SAT techniques since it is rather a feasibility problem (one variable in the objective function) than an optimization problem. Computational results on standard benchmark instances reveal this fact, while we also show in this thesis that depending on some problem characteristics, the choice of the algorithm varies. The best results in the area of CP and SAT solving are given by the works of Schutt et al. [225, 227] with a CP-SAT hybrid (CPSAT), Horbach [144] with a pure SAT model (SAT) and Liess and Michelon [177] using a pure CP approach (CP). On the Pack instances, the best results have been generated by Carlier and Néron [43] by introducing redundant resources and using energetic reasoning as propagation algorithm. Hence, we report these values for the CP. Table 1.5 summarizes the capabilities of these solvers to solve instances from the different test sets revealing the strength of SAT based hybrid solvers.

Table 1.5: Comparison of exact procedures based on CP and SAT by comparing the percentage of optimally solved instances. '–' indicates that no values are reported.

	J30	J60	J90	J120	Pack
CP	97.7%	81.2%	78.8%	40.0%	80.0%
SAT	97.3%	84.0%	79.0%	41.2%	–
CPSAT	100.0%	89.6%	82.7%	47.0%	69.1%

Certainly, all techniques, from propagation algorithms, over the way conflict analysis is implemented, to the model used, computers, time limit and so on, play a crucial role when comparing these instances. Some procedures turned out to be well-suited for small but not for medium or large instances. Hence, we did not reimplement these, but mention them next. In general, the cut-set rule of Demeulemeester and Herroelen [84] is most efficient for small size instances. It is based on storing partial schedules. The local bounds of variables are checked for being dominated by these partial schedules. Due to the high number of cut-sets to be stored it does not scale to medium or large size instances. Good results are reported for instances with up to 50 jobs, see [84, 177]. Another often well-suited technique is to branch by posting new precedence constraints. Laborie [168] concludes that branching by posting new precedence constraints for each minimal conflicting set is too expensive for highly cumulative instances. In our experiments, this kind of branching has an even more negative impact as our solver does not support local constraints in conflict clauses – only variables are allowed. This technique is not better suited in our solver. Last, we point out that several IP formulations exist and could be used. We restrict to the one of Pritsker et al. [209] that is well suited for a CIP solver like SCIP due to the knapsack like resource constraints and a simple linking constraint. Nevertheless, different models might prove useful on different instances. In this thesis, we concentrate on the integration of CP, IP ad SAT techniques in a CIP environment. Comparisons between available IP formulations can be found in [12, 164].

1.5.3 Setup for experiments

In this thesis we compare the solvers and the used propagation, separation and explanation algorithms with each other. Benchmark instances are available, but some of them are easily solved or too hard to solve. We concretize next how all experiments are done in order to get comparable results within reasonable time.

Environment To perform our experimental study we use the non-commercial Constraint Integer Programming framework SCIP [2], version 2.1.1 with bug-fixes and some refinements that make e.g., bound-widening in conflict analysis possible. We integrated CPLEX release version 12.4 as underlying LP solver. All computations reported are obtained on Dual QuadCore Xeon X5550 2.67 GHz computers (in 64 bit mode), 24 GB of main memory, running a Linux system using GNU compiler 4.6.2.

SCIP has a SAT-like conflict analysis mechanism and is a backtracking system. To avoid an overhead by constructing explanations for bound changes, it is possible to store additional information for each bound change. Since the number of stored bound changes is quite large during the search, the space for these information are restricted to 32 bits each. In the corresponding section, we explain how we use these bits to deliver explanations efficiently.

We turn off all heuristics during branch-and-bound search since global primal heuristics sometimes find feasible solutions by different branching decisions. Prior to branch-and-bound search, we use a scheduling-specific serial SGS based on the a sorting of the start times variables that is set up according to a relaxed problem (LP solution or CP solution), see [188]. These solutions are in most cases not optimal and are only improved during search if an earliest start schedule that respects the local bounds yields a feasible solution.

Instances To measure the impact of solving techniques on instances from RCPSP, we concentrate on instances from PSPLib and on the Pack instances. in order to test parameter settings within reasonable running time, we shrink the test set instead of lowering the time limit. In order to shrink the size of the test set we collect a subset of the instances where different behaviors of the solving techniques can be observed. We carried out an initial study with default CP settings and erased all instances that have been either easily solved or too hard to solve within one hour. For all test sets we used the following criteria to restrict the test set to reasonable instances. From the PSPLib we kept all instances which:

1. could be solved to optimality by at least one solver,
2. at least one solver needed more than one search node, and
3. at least one solver needed more than one second of running time.

Then, we grouped these instances into small ones ("setS"), subsets from J30 and J60, and large ones ("setL"), subsets from sets J90 and J120. We kept the whole Pack set consisting of 55 instances, though several are too hard to solve and only on 28 of them the criteria would apply.

We list the set of instances here: From set J30, we took the instances 5_{2,4}, 9_{1–10}, 13_{1–10}, 14_{2,7}, 25_{1–9}, 29_{1–10}, 30_{2,3}, 37_7, 41_{1–10}. 45_{1–10} and 46_7. From set J60, 5_{1–9}, 9_4, 14_4, 17_8, 21_{1–10}, 26_{3,4,9}, 37_{1–10}, 38_2, 41_{4,7}, 42_{3,4,7,8} and 46_{1,4,5,6,7,8}. From set J90 we are left with 1_{1–10}, 5_{1,2}, 6_8, 17_{1,2,3,8,9,10}, 21_{3,4}, 26_{4,7}, 33_{1,5,9,10}, 37_{3,4,7,9}, 42_9 and 46_{1,3,5}. From set J120 the following instances remain: 1_{2,4–7,9,10}, 2_{1–10}, 3_1, 8_1, 14_{3,9}, 15_3, 19_1, 21_{3–6,8–10}, 22_{1,2,4–10}, 23_{5,6,9}, 28_{2,4,6}, 29_{4,5,8}, 30_{5,7}, 34_{4,6,7}, 35_9, 41_{1–8,10}, 42_{3–10}, 43_{1,3–6,9}, 48_9, 49_{5,7,8,9} and 50_{2,10}

For the overall performance, we finally report results on the set of all instances. Later, for applications of our techniques we use additional test sets for labor constrained scheduling problems, for RCPSP with discounted cash flows and for a multi-mode resource leveling problem that will be described in the corresponding chapters.

Tables and figures In order to compare different settings or different solvers with each other we will show elaborate results in tables such as given in Table 1.6. These tables contain the number of optimally solved instances ('nopt'), how often which solver found the best primal ('bprimal') or best dual ('bdual') bound. The average gap of all instances from the test set is given in column 'gap', while the gap after solving the root relaxation is given in column 'gapRoot'. Gaps are computed for a primal bound p and dual bound d via $(p - d)/d$. Choosing d as denominator instead of p, we report a more pessimistic final gap. When comparing presolving and lower bounding techniques, we also show the gap between the final dual bound d_f and the root dual bound d_r given by $(d_f - d_r)/d_r$ which measures by how much the dual bound has been improved. The gaps are shown in percentages, while in later tables we do not display the %-sign.

On those instances that are solved to optimality by all solvers, we compute the average running time ('avtime') in seconds and the average number of nodes ('avnodes') needed by each solver on this restricted subset. The cardinality of this subset is displayed after 'allopt:'.

setting	nopt	bprimal	bdual	gapRoot[%]	gap[%]	avtime [sec]	avnodes [sec]
setS (114 instances)						(allopt: e.g., 19 instances)	

Table 1.6: Headers of the tables used in the experiments.

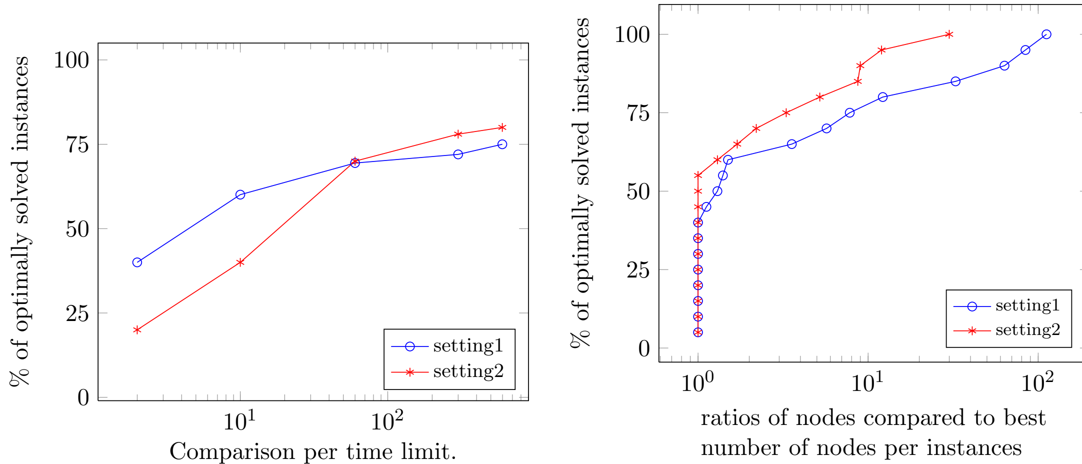


Figure 1.2: On the left, the number of optimally solved instances per time limit is shown, while on the right, a distribution function is given that shows which multiple of nodes (or running time) is needed for a certain setting to solve an instance compared to a best setting.

On the picked test instances from setS, setL and Pack, the number of nodes and the running time vary widely between 100 and $500,000$ nodes. Hence, we do not compute the standard deviations from the average values as these are always pretty high. Instead, we support visualizing the impact of the different settings on the number of nodes and on the running time with the help of distribution functions. In Figure 1.2 on the left, the distribution function for different time limits is given. The time limits on the horizontal axis are chosen to be $2, 10, 60, 300$ and 600 seconds in logarithmic scale. On the vertical axis, the percentage of the instances that are solved to optimality are shown. Hence, each marked point corresponds to the percentage of the instances that can be solved within the given time limit. Connecting these points linearly is done to support visualization per setting. Points on the lines do not correspond to the percentage of instances that can be solved within intermediate time limits. On the right of Figure 1.2, a distribution function is shown. On the vertical axis, we again show the percentage of optimally solved instances. For each instance solved to optimality, we compute the minimum number of nodes needed to solve this instance over all the considered settings. Then, a ratio per setting is computed as the number of nodes needed by this setting divided by the minimum number of nodes any solver needed on this instance. The functions in the figure now show the distribution function of these ratios in logarithmic scale. Hence, a point on a line of this curve corresponds to the percentage of instances that can be solved by needing at most a certain multiple of the minimum number of nodes. A similar distribution function will be shown for the running times on all optimally solved instances.

Chapter 2

CIP Techniques for RCPSP

In this chapter we study and develop solving techniques in a Constraint Integer Programming (CIP) framework for RCPSP. Recall the CIP formulation for RCPSP from Chapter 1:

$$\begin{aligned} \min \quad & C_{\max} & \\ \text{subject to} \quad & \texttt{precedence}(S_i, S_j, p_i) & \forall\, (i,j) \in E \\ & \texttt{cumulative}(\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}_{.k}, R_k) & \forall\, k \in \mathcal{R} \\ & D(S_j) = \mathbb{N}_0 & \forall\, j \in \mathcal{J}. \end{aligned}$$

In a CIP framework, the different techniques from Integer Programming (use of LP relaxation, pseudo-cost branching, LP-based heuristics,...), Constraint Programming (domain propagation, inference or constraint branching,...) and SAT (conflict analysis, hot restarts, conflict-driven search) are used throughout a branch-and-bound search. The techniques are either implemented in the constraints which provide callback methods or by other propagators or separators that are part of the framework.

Within this framework we evaluate the impact of the different solving techniques on the solution process.

Nowadays, hybrid approaches that combine CP and SAT techniques like [137, 228], prove most successful in order to solve hard scheduling instances efficiently. We show the importance of the IP part in such hybrid frameworks by presenting a *continuous relaxation* of the cumulative constraint. We study algorithms and their complexity for propagation, explanation and separation procedures. In contrast to previous literature on explanation algorithms for cumulative scheduling, we carry out a complete complexity study on optimal explanation algorithms for the propagation algorithms time-tabling, edge-finding time-tabling edge-finding and energetic reasoning. We develop different explanation algorithms for these propagation rules and show in a computational study the merits of minimum size explanations in contrast to less elaborate approaches and in contrast to a pure CP search without conflict analysis.

Contribution After discussing the main propagation algorithms for the cumulative constraint, we develop techniques to speed up the energetic reasoning propagation algorithm. We show that using an approximative criterion to estimate the energy of an interval, we are able to improve the running times on PSPLib instances to 1/4 if energetic reasoning is used as stand-alone propagation algorithm. Nevertheless, on most instances it is best to use the time-tabling algorithm.

We study and develop explanation algorithms for bound adjustments detected by the cumulative propagation algorithms. To this end, we introduce the notion of *explanations*, which we show to be not unique in case of the cumulative constraint, in contrast to standard clauses in SAT. Using our sophisticated explanation algorithms, we are able to reduce the average number of branching nodes by 90% on instances from PSPLib, which is remarkable. As a computational overhead is induced by performing conflict analysis, the average running times will only decrease to 33%. Interestingly, on the Pack instances these good results are not obtained, which may be due to the nature of these highly cumulative instances because of which time-tabling is not able to detect bound adjustments.

The use of a continuous relaxation enables the hybrid solver to apply MIP techniques, in particular branching on fractional values and to use pseudocost branching to guide the branching decisions. Using this technique, we are able to solve few more instances from each test set while the number of nodes decreases by one-half and the average running time decreases to two-third in contrast to the best CP-SAT hybrid developed before.

Outline Section 2.1.2 presents basic constraints, like `precedence` (that models the precedence relations in scheduling), `disjunctive` (that models jobs that are not allowed to be executed in parallel) and `bounddisjunction` (that models constraints that result from conflict analysis). Then, in Section 2.1.3 the CP techniques for the `cumulative` constraint are presented. We develop an approximative criterion for energetic reasoning to detect an overload in Section 2.1.4. This criterion is used in a parametrical propagation algorithm. In Section 2.2 we describe hybridizations of CP, IP and SAT and in particular the process of conflict analysis for general constraints based on an example from cumulative scheduling. We invest a theoretical and computational study on how to best perform conflict analysis for various explanation algorithms. In Section 2.3.1 we study the complexity of delivering optimal (minimum size and minimum weight) explanations to these algorithms. We develop explanation algorithms that differ in strength and running time in Section 2.3.2. We conclude with an experimental study and compare the developed explanation algorithms on standard benchmark instances from PSPLib [162]. Section 2.4 presents a *continuous relaxation* of the cumulative constraint. This relaxation is only based on the integer start time variables and has to the best of our knowledge not been used successfully for RCPSP so far.

2.1 CP techniques

CP is famous for its better scalability on instances from RCPSP in contrast to MIP because no time-indexed variables need to be introduced. Furthermore, RCPSP can be solved via a binary search on the makespan after which a pure satisfiability problem remains. Also because of the high logical structure in such instances, CP techniques have been successfully applied to solve series of hard scheduling problems.

In this section, we start by introducing *constraint propagation* and the notation for bound changes needed in our study on explanation algorithms in Section 2.2. Then, several well-known propagation algorithms for scheduling problems are discussed in Section 2.1.2. In particular, in Section 2.1.3 we concentrate on propagation algorithms for the cumulative constraint and compare their impact when solving standard benchmark instances. This study shows, that time-tabling is best to be used on instances from

PSPLib, while energetic algorithms such as edge-finding or time-table edge-finding are very useful on highly cumulative Pack instances.

Section 2.1.4 is devoted to the energetic reasoning algorithm for which we propose an approximative criterion to decide whether too execute this powerful but costly algorithm. On instances from PSPLib, the running times can be decreased to 1/4 on average, whereas on highly cumulative instances this technique does not yield any improvement.

2.1.1 Constraint propagation

Recall from Section 1.3.2, that constraint propagation is a technique to reduce the search space by excluding inconsistent values from the domains of the variables. We start by introducing the basic notation for this technique, see e.g., [1]. With each continuous variable S_j with earliest start time $\mathrm{est}_j \geq 0$ and latest start time $\mathrm{lst}_j \geq \mathrm{est}_j$ we associate its domain $D_j := [\mathrm{est}_j, \ldots, \mathrm{lst}_j]$. We abuse notation by writing $S_j \in D_j$ for the assignment of variable S_j. It becomes clear from the context whether a variable or its assignment is considered. Using a MIP notation based on halfspaces, we can express the domain of a variable by $D_j = \{S_j \geq \mathrm{est}_j\} \cap \{S_j \leq \mathrm{lst}_j\}$.

Throughout branch-and-bound the domains are reduced by branching decisions or by propagation algorithms. A halfspace that reduces the domain of a variable is written in the form $\{S_j \geq \mu^L\}$, resp. $\{S_j \leq \mu^U\}$, where $\mu^L > \mathrm{est}_j$, resp. $\mu^U < \mathrm{lst}_j$. Let $\mathcal{D} = D_1 \times \ldots \times D_n$ be the vector of domains per variable.

Given a node of branch-and-bound tree, we consider the unique path from the root to that node and denote by $\mathcal{B} = \{B_1, \ldots B_k\}$ the set of all so far performed domain reductions (branching decisions and deductions from propagation algorithms). More formally, we denote the set of all lower bound changes of variable S_j by $\boldsymbol{L}^j := \{\boldsymbol{L}_1^j, \ldots, \boldsymbol{L}_{L_j}^j\}$, where for $k = 1, \ldots, L_j$ the sets $\boldsymbol{L}_k^j$ denote the halfspaces $\boldsymbol{L}_k^j = \{S_j \geq \mu_k^L\}$. Equivalently, we denote the set of all upper bound changes by $\boldsymbol{U}^j := \{\boldsymbol{U}_1^j, \ldots, \boldsymbol{U}_{U_j}^j\}$, where $\boldsymbol{U}_k^j = \{S_j \leq \mu_k^U\}$. Then, $\mathcal{D} = \bigcap_{j \in \mathcal{J}} (\bigcap_{i=1}^{L_j} \boldsymbol{L}_k^j \cap \bigcap_{i=1}^{U_j} \boldsymbol{U}_k^j) = \bigcap_{\ell=1}^{k} B_\ell$.

Now, we formalize our notation of a propagation algorithm. A propagation algorithm P gets as input a set S of variables, the domains $\mathcal{D}$ represented via halfspaces $\mathcal{B}$ and one or more constraints $C(S)$ over the variables. It is more convenient in the next definition to use $\mathcal{B}$ instead of $\mathcal{D}$ to describe the local bounds of the variables, as these are used in our study on explanations for the propagation algorithms. A propagation algorithm uses the logical implications of the given constraint(s) to perform domain reductions, also equivalently referred to as *bound adjustments*, *bound changes* or, in MIP notation, as halfspaces. Adjusting the lower bound of variable S_j to some value of at least est'_j is denoted by the halfspace $\{S_j \geq \mathrm{est}'_j\}$. Propagation algorithms only perform feasible bound adjustments, i.e., the set of feasible solutions before and after the domain reduction remains the same[1]. The set of all feasible domain reductions (halfspaces) by propagation algorithm P under constraint $C(S)$ and bounds $\mathcal{B}$ is denoted by $f_P(S, \mathcal{B}, C(S))$. Hence, we write $\{S_j \geq \mathrm{est}'_j\} \in f_P(S, \mathcal{B}, C(S))$. This notation will be used in our study on explanation algorithms in Section 2.2. In the sequel of this section, where we present the bound adjustments of different propagation algorithms, we state the values est'_j in the corresponding lemmas, instead of using the halfspace notation.

[1] A weaker assumption can be used in MIP (CP): at least one optimal (feasible) solution must remain if the solution space is non-empty.

Example 2.1. We consider the network with seven jobs as depicted in Figure 2.1. The processing times and demands are shown in the table of Figure 2.2. The resource capacity is three ($R = 3$). Figure 2.1 shows that a feasible solution with makespan 13 exists. Assume that a hypothetical makespan of $C_{\max} = 14$ is known. Column 4 of Figure 2.2 shows the global bounds of the variables due to precedence constraints with a global makespan of 14. Columns 5, 6 and 7 show the variable bounds after iteratively applying the branching decisions $\{S_B \geq 1\}$, $\{S_G \leq 5\}$ and $\{S_E \leq 8\}$ with propagation of the precedence and resource constraints using time-tabling propagation as described in Section 2.1.3.2. After the last branching decision has been propagated, job A can be shifted over its latest start time and therefore the problem becomes infeasible. This example will be used later in our study on explanations for infeasibilities and bound changes.

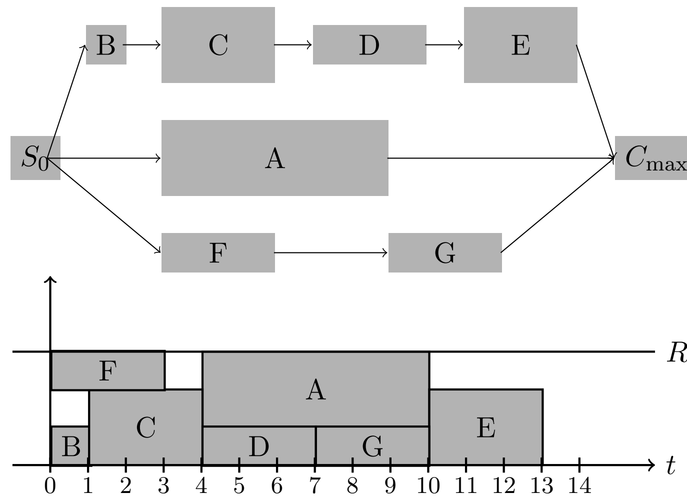


Figure 2.1: A precedence network of seven jobs $A, B, \ldots, G$ and two dummy jobs (start S_0 and end $C_{\max}$) and an optimal solution with makespan $C_{\max} = 13$ are depicted.

2.1.2 Propagation of basic constraints

Propagation algorithms are the main algorithms in CP solvers. They are applied more or less in every node of the search tree in order to detect inconsistent variable assignments and to remove these from the search space. Next, we recall the constraints and their propagation algorithms as they are needed for resource-constrained project scheduling problems. These constraints are the *precedence constraints*, the *disjunctive constraints* and the *bounddisjunction constraints*. The former are needed in conflict analysis.

2.1.2.1 The precedence constraint

A *precedence constraint* $i \prec j$ models the relation between two jobs i and j. It enforces that in a precedence-feasible solution $S_i + p_i \leq S_j$ holds.

Precedence constraints do not only occur in scheduling problems. They are part of many IP and CP formulations and binary or continuous variables can be involved and scaling factors may appear. Hence, in CP solvers the more general `varbound` constraint exists, whose input are variables x, y (binary, continuous or integer) and parameters

	p_j	r_j	global bounds $[\text{est}_j; \text{lst}_j]$	1st branch $[\text{est}_j; \text{lst}_j]$	2nd branch $[\text{est}_j; \text{lst}_j]$	3rd branch $[\text{est}_j; \text{lst}_j]$
A	6	2	$[0; 8]$	$[0; 8]$	$[0; 8]$	↯
B	1	1	$[0; 4]$	$[1; 4]$	$[1; 4]$	$[1; 1]$
C	3	2	$[1; 5]$	$[2; 5]$	$[2; 5]$	$[2; 2]$
D	3	1	$[4; 8]$	$[5; 8]$	$[5; 8]$	$[5; 5]$
E	3	2	$[7; 11]$	$[8; 11]$	$[8; 11]$	$[8; 8]$
F	3	1	$[0; 8]$	$[0; 8]$	$[0; 2]$	$[0; 2]$
G	3	1	$[3; 11]$	$[3; 11]$	$[3; 5]$	$[3; 5]$
$C_{\max}$	0	0	$[10; 14]$	$[11; 14]$	$[11; 14]$	$[11; 14]$

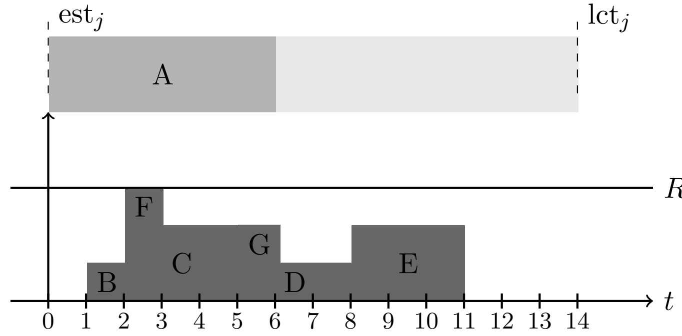


Figure 2.2: On top, the variable bounds before and after branching are displayed. Below, a time-table after the third branching decision is shown. Using time-tabling propagation, the lower bound of job A can be updated from zero to eleven, thereby violating the latest start time $\text{lst}_a = 8$.

$\alpha, a, b \in \mathbb{Q}$.

$$\texttt{varbound}(x, y, \alpha, a, b): \qquad \{(x, y) \in (D(x), D(y)) \mid a \leq \alpha \cdot x + y \leq b\}.$$

Since our approach will not need these more general constraints, we stick to the `precedence`-constraint defined for integer variables S_i, S_j and a parameter p_i:

$$\texttt{precedence}(S_i, S_j, p_i): \qquad \{(S_i, S_j) \in (D(S_i), D(S_j)) \mid S_i + p_i \leq S_j\}.$$

The following bound changes can be derived:

Lemma 2.1. *Given a* `precedence`(S_i, S_j, p_i)*-constraint.*
(i) A valid lower bound est'_j *on the start time* S_j *of job* j *with respect to this constraint is given by:*

$$\text{est}'_j = \max\{\text{est}_j, \text{est}_i + p_i\}.$$

(ii) A valid upper bound lst'_i *on the start time* S_i *of job* i *with respect to this constraint is given by:*

$$\text{lst}'_i = \min\{\text{lst}_i, \text{lst}_j - p_i\}.$$

Given that observation, we may deduce an infeasibility:

Lemma 2.2. *The problem is infeasible w.r.t.* `precedence`(S_i, S_j, p_i) *if* $\text{est}_i + p_i > \text{lst}_j$ *holds.*

Propagating precedence constraints has been done for a long time in project scheduling. For problems containing minimum and maximum time-lags, a Bellman or a Floyd Warshall algorithm with a complexity of $O(n^3)$ is used. For acyclic precedence networks a topological sorting of the jobs is used such that propagating these constraints runs in linear time $O(m)$.

2.1.2.2 The disjunctive constraint

In many applications, some jobs are not allowed to be executed in parallel due to technical restrictions or due to a limited number of resources. An extreme case is given if all jobs cannot be executed in parallel. Then, we speak of single machine scheduling problems. If only a subset of jobs cannot be executed in parallel and there is no transitivity given, then such scheduling problems are referred to as *scheduling with conflicts* [108, 148].

From a CIP perspective we will use the `disjunctive` constraint to model the conflicts between pairs of jobs $i, j \in \mathcal{J}$ with $i \neq j$:

$$\texttt{disjunctive}(S_i, S_j, p_i, p_j) :$$
$$\{(S_i, S_j) \in (D(S_i), D(S_j)) \mid S_i + p_i \leq S_j \dot{\vee} S_j + p_j \leq S_i\}$$

An IP formulation for this constraint is obtained by introducing a binary variable x_{ij} that models whether job i precedes job j ($x_{ij} = 1$) or whether job j precedes job i ($x_{ij} = 0$) and the following big-M-constraints for some large value M:

$$S_i + p_i \leq (1 - x_{ij}) \cdot M + S_j \qquad \wedge \qquad S_j + p_j \leq x_{ij} \cdot M + S_i.$$

Here, a pair of jobs is not allowed to be executed in parallel. Observe that this must not be a transitive relation between the jobs. An extreme case can be found in single-machine or sequencing problems, where no job is allowed to be executed in parallel to any other. Cumulative constraints (see next section) with capacity one are another extreme case, where all jobs from a single constraint are not allowed to be executed in parallel.

Due to the huge amount of variables (in the worst case quadratic in the number of jobs) that might be introduced and the weak dual bounds due to the bigM-formulation, we do not linearize this constraint but use it throughout branch-and-bound for propagation.

It is easy to check for each `disjunctive` constraint whether the variable bounds are already that tight, such that the constraint can be replaced by a `precedence` constraint, as the following lemma summarizes.

Lemma 2.3. *Given a disjunctive pair* (i, j)*.*
(i) If $\text{lct}_i < \text{ect}_j$*, then job* i *precedes job* j*.*
(ii) If $\text{lct}_j < \text{ect}_i$*, then job* j *precedes job* i*.*

Instead of posting new precedence constraints locally in a node of the search tree, the domains can be propagated according to Lemma 2.1.

Lemma 2.4. *If for a disjunctive pair* (i, j) $\text{lct}_i < \text{ect}_j$ *holds then* $\text{est}'_j \geq \text{ect}_i$ *and* $\text{lst}'_i \leq \text{lst}_j - p_i$*.*

Infeasibilities are detected if both domains are that tight that the jobs must be scheduled in parallel. We denote by $\text{lct}_{\{i,j\}} := \max\{\text{lct}_i, \text{lct}_j\}$ and by $\text{est}_{\{i,j\}} := \min\{\text{est}_i, \text{est}_j\}$.

Lemma 2.5. *Given a disjunctive pair (i,j). If $\mathrm{lct}_{\{i,j\}} - \mathrm{est}_{\{i,j\}} < p_i + p_j$, then the problem is infeasible.*

If a problem consists of resources with unit capacity, more specialized algorithms than for general capacity values can be applied, see e.g., [92]. In Section 3.2.2 we will use the set of all pairs of disjunctive jobs to identify large cliques and build redundant cumulative resource constraints with capacity one.

2.1.2.3 The bounddisjunction constraint

The `bounddisjunction` constraint [1] is part of the SCIP framework. We introduce it here shortly because it is created throughout branch-and-bound as a result of conflict analysis. The `bounddisjunction` constraints are conflict constraints (no-goods) for integer or continuous variables. Given an n-dimensional vector x and an m-dimensional vector y of variables and two vectors $a \in \mathbb{Q}^n, b \in \mathbb{Q}^m$, the `bounddisjunction` constraint is defined as:

$$\texttt{bounddisjunction}(x,y,a,b): \left\{ x_i \in D(x_i), y_i \in D(y_i) \mid \bigcup_{i=1}^{n}\{x_i \geq a_i\} \cup \bigcup_{i=1}^{m}\{y_i \leq b_i\} \right\}.$$

This constraint can only propagate if at most one variable does not obey its bound condition. Hence, it is expensive to propagate them if many of them are present or if a single constraint contains many variables. From a computational point of view it is good to have few `bounddisjunction` constraints in the model, and each with few variables. The initial SCIP settings only keep these constraints from conflict analysis if at most 8 variables or not more than 10% of the variables are involved in the generated conflict constraints. In case of scheduling we increase this number to at least 10 or 10% of the variables.

2.1.3 The cumulative constraint

The resource constraints play an important role, since they turn RCPSP into an $\mathcal{NP}$-hard problem. First, we give an overview on the most important domain propagation techniques for the cumulative constraint, also called *propagators*. We will only present the basic ideas of the algorithms we use and formally state the conditions under which problems are detected to be infeasible and the conditions to perform bound changes. These basics are needed in Section 2.1.4 where we find an approximative criterion fora propagation algorithm, called *energetic reasoning*, to detect infeasibilities and in Section 2.2 where we develop explanation algorithms for the different propagators.

In this section, we consider exactly one particular cumulative constraint and recall the notion omitting the index k of the resource. In cumulative scheduling, an instance is given by a set $\mathcal{J} = \{1,\ldots,n\}$ of non-preemptable jobs with processing times $p_j \in \mathbb{N}$ for each job $j \in \mathcal{J}$. Each job j requires a certain number r_j of a cumulative resource which has a capacity $R \in \mathbb{N}$. In a constraint program, a *cumulative* constraint as introduced in [5] is given by $\texttt{cumulative}(\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}, R)$, defined by vectors of start times $\boldsymbol{S} = (S_1,\ldots,S_n)$, processing times $\boldsymbol{p} = (p_1,\ldots,p_n)$ and resource requests $\boldsymbol{r} = (r_1,\ldots,r_n)$,

and the capacity R. The cumulative constraint enforces that at each point in time t, the cumulative demand of all jobs running at t, does not exceed the given capacity, i.e.,

$$\sum_{j \in \mathcal{J}: t \in [S_j, S_j + p_j)} r_j \leq R \qquad \text{for all } t.$$

Depending on the *earliest start times* (est_j), *earliest completion times* (ect_j), *latest start times* (lst_j), and *latest completion times* (lct_j) for each job $j \in \mathcal{J}$, propagation algorithms update variable bounds. The domain of a start time variable S_j corresponds to the interval $[\text{est}_j; \text{lst}_j]$. The *feasible interval* when a job can be processed is given by $[\text{est}_j; \text{lct}_j)$.

2.1.3.1 Checking resource feasibility of a solution

Given some solution to RCPSP or for other cumulative scheduling problems. It is easily checked whether precedence constraints are fulfilled, while for checking resource-feasibility more efforts are needed. In the literature, see e.g. [12], a procedure with quadratic running time $O(n^2)$ is given that checks for one resource constraint whether it is violated by a solution or not. The authors sort all jobs by non-decreasing completion time and check at each such point in time whether the jobs that are running or complete at this point in time violate the capacity. As there are possibly n points in time and $O(n)$ jobs to be considered, this yields a total running time of $O(n^2)$. It is possible to speed up their procedure to run in $O(n \log n)$. For completeness, we state Algorithm 1 with that efficiency here. The idea is to keep two sorted lists L_1, L_2, for which indeed two arrays can be used. Observe that the resource profile changes only at event points that correspond to the start or completion of a job. The first list L_1 is sorted by non-decreasing start times $S_j^\star$ and the second one by non-decreasing completion times $S_j^\star + p_j$. Simultaneously to sorting both lists, we keep track which resource demands belong to the designated start and completion times. Then, we run through both lists in a synchronized manner by using one counter per list for fast access. According to that ordering, each event (at most $2n$) is checked whether the capacity is obeyed or not. For that purpose, all resource demands that complete before that point in time, are subtracted and all resource demands of starting jobs are added. If the capacity is exceeded, the problem is infeasible. In contrast to the approaches from the literature, we need to sort two lists, keep track of the associated resource demands for each point in time and need to consider $2n$ points in time instead of n. Altogether, this yields an $O(n \log n)$ algorithm.

2.1.3.2 Propagation algorithms

Several CP techniques focus on propagation algorithms for the cumulative constraint, see e.g. [12, 22, 45, 95, 159, 230, 263]. We give a survey of the basic ideas of the propagation algorithms time-tabling, edge-finding, time-tabling edge-finding and energetic reasoning.

We only consider the updates of the lower bounds since updating the latest completion times can be handled symmetrically. As introduced before, bound changes are halfspaces induced on a single variable, e.g., $\{S_j \geq \text{est}_j\}$ for some $j \in \mathcal{J}$. We clarify how the propagation algorithms update the variable bounds in separate lemmas. If a propagation algorithm updates the bound of variable S_j from est_j to some larger value est'_j, this is equivalent to adding a halfspace $\{S_j \geq \text{est}'_j\}$, a notation used for explanation algorithms.

Algorithm 1: Checking feasibility of a solution for one cumulative constraint.

Input: Resource capacity R, set $\mathcal{J}$ of jobs and their schedule $S^\star$.
Output: Returns whether solution is feasible or not.

1. Create list of points in time L_1 sorted by $S^\star_j$.
2. Create list of points in time L_2 sorted by $S^\star_j + p_j$.
3. Set $D = 0$.
4. **for** $t \in L_1 \cup L_2$ *in non-decreasing order* **do**
5. Subtract demands of jobs that end until t from D.
6. Add demands of jobs that start at t to D.
7. **if** $D > R$ **then**
8. **return** *False*.
9. **return** *True*.

Time-tabling *Time-tabling* is also known as the concept of *core-times* [159]. The *core* of a job is defined by the interval $\gamma_j := [\text{lst}_j, \text{ect}_j)$. Observe that a core can be empty. Intuitively, this is the interval in which the job must be processed due to its earliest start and latest completion time since preemption is not allowed. We define the *core-profile* $\Gamma_{\mathcal{J}} : t \to \mathbb{N}$ for a set of tasks $\mathcal{J}$ as:

$$\Gamma_{\mathcal{J}}(t) := \sum_{j:t\in\gamma_j} r_j.$$

This profile can be computed in $O(n\log(n))$ by first sorting the jobs according to the start and completion times of the cores which yield at most $2n$ event points to be considered. Then, the profile is created in the order of these events. A point in time t at which the capacity is exceeded, i.e., $\Gamma_{\mathcal{J}}(t) > R$, is called a *peak*.

If due to their cores too many jobs need to be executed at a particular point in time t, then the problem is infeasible. We summarize this in the following lemma.

Lemma 2.6. *The problem is infeasible if* $\Gamma(t) > R$ *holds for some* t.

Now, we turn to the time-tabling propagation algorithm. A job j can be scheduled at time t if $\Gamma_{\mathcal{J}\setminus\{j\}}(\tau) \le R - r_j$ holds for all $\tau = t, \ldots, t + p_j - 1$. Bound adjustments can be made by trying to insert each job j as early as possible into the core-profile $\Gamma_{\mathcal{J}\setminus\{j\}}$. If it cannot be scheduled there, the earliest time slot of size p_j where the job can be scheduled yields a new lower bound on the start time S_j.

Lemma 2.7. *A valid lower bound* est'_j *on the start time* S_j *of job* j *with respect to the resource profile is given by:*

$$\text{est}'_j := \min\left\{t \ge \text{est}_j \mid \Gamma_{\mathcal{J}\setminus\{j\}}(\tau) \le R - r_j \text{ for } \tau = t, ..., t + p_j - 1\right\}. \tag{2.1}$$

A job may be shifted over several peaks $R - r_j$ to some larger value (even larger than lst_j) in one iteration. If the job admits a core itself after updating the bound, this core can be immediately inserted into the core-profile in time $O(\log(n) + p_j)$. Hence, time-tabling can be easily implemented as a dynamic propagation algorithm (i.e., an algorithm that performs bound changes immediately and updates the core-profile in every iteration), see Algorithm 2. We remark that in line 4 of Algorithm 2 any order can

Algorithm 2: Implementation of time-tabling.

Input: Resource capacity R, set $\mathcal{J}$ of jobs.

Output: Earliest start times est'_j for each job j or an infeasibility is detected.

1 Create profile $\Gamma_{\mathcal{J}}$.
2 **if** $\exists\, t : \Gamma_{\mathcal{J}}(t) > R$ **then**
3   **return** Infeasible.
4 **foreach** *job j in order of non-decreasing* est_j **do**
5   **if** *job j can be scheduled at* est_j *in* $\Gamma_{\mathcal{J}\setminus\{j\}}$ **then**
6   continue.
7   Find est'_j according to $\Gamma_{\mathcal{J}\setminus\{j\}}$ by scanning the profile from left to right.
8   **if** $\mathrm{est}'_j > \mathrm{lst}_j$ **then**
9   **return** Infeasible.
10   Set $\mathrm{est}_j := \mathrm{est}'_j$, possibly update $\Gamma_{\mathcal{J}}$.

be chosen, but an ordering according to earliest start times is computationally beneficial as after an update new cores may occur or the cores just expand.

Example 2.2. Figure 2.3 visualizes how the core of a single job is computed. On the right, two jobs each with demand $r_1 = r_2 = 2$ and processing times $p_1 = 4$, and $p_2 = 3$ and processing intervals $[\mathrm{est}_1, \mathrm{lct}_1) = [0, 6)$, and $[\mathrm{est}_2, \mathrm{lct}_2) = [0, 4)$, resp., are given. There is a peak at $t = 2$.

In Figure 2.4, the core-profile for some set of jobs is depicted. The core of the specified job j is empty, but job j cannot be scheduled at its lower bound est_j, since at $t = 3$ the core-profile exceeds $R - r_j$. Hence, $\mathrm{est}'_j = 4$. Furthermore, at $t = 5 \in [\mathrm{est}'_j, \mathrm{est}'_j + p_j)$ again a peak larger than $R - r_j$ exists. Repeating this procedure, an update to $\mathrm{est}'_j = 13$ can be propagated.

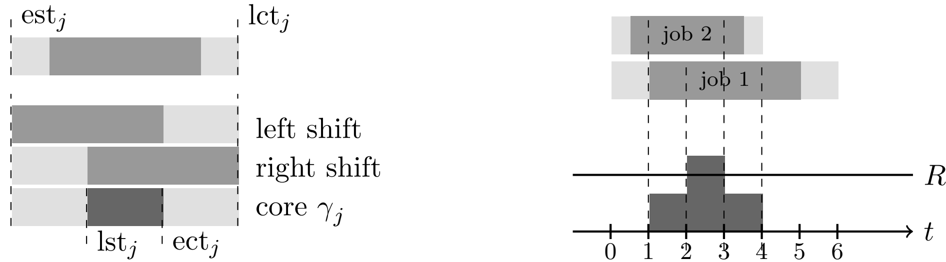


Figure 2.3: On the left: Visualization of a core γ_j of job j. On the right: The core-profile for two jobs where the capacity R is exceeded in the interval $[2; 3)$.

Energetic reasoning The *energy* of a job is the product of the processing time and the resource demand. Energetic reasoning checks non-empty time intervals $[a, b)$, with $a < b$, whether all jobs contributing to that interval require more energy than available. This concept is also known as *interval consistency test*, see e.g. [95]. In a problem setting of n jobs there are $O(n^2)$ intervals to be checked for feasibility, see Baptiste et al. [22]. These

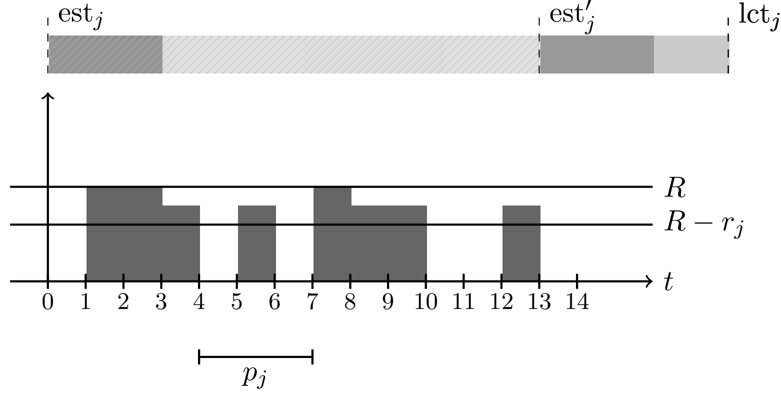


Figure 2.4: Illustration of time-tabling propagation. The lower bound of job j can be updated from zero to 13 due to the cores that exceed $R - r_j$ at least every p_j time units before $t = 13$.

intervals can be computed from the earliest and latest start times. The *available energy* of such an interval $[a, b)$ is given by $R \cdot (b - a)$. The required energy $e_j^{\mathrm{ER}}(a, b)$ of job j in interval $[a, b)$ is defined by:

$$e_j^{\mathrm{ER}}(a, b) := \max\{0, \min\{b - a, p_j, \mathrm{ect}_j - a, b - \mathrm{lst}_j\}\} \cdot r_j. \tag{2.2}$$

Hence, $e_j^{\mathrm{ER}}(a, b)$ is the non-negative minimum of (i) the required energy if it runs completely in the interval $[a, b)$, i.e., $(b-a) \cdot r_j$, (ii) the required energy of job j, i.e., $p_j \cdot r_j$, (iii) the left-shifted energy, i.e., $(\mathrm{ect}_j - a) \cdot r_j$, and (iv) the right-shifted energy, i.e., $(b - \mathrm{lst}_j) \cdot r_j$. Figure 2.5 sketches the different situations how an interval $[a, b)$ can be positioned towards a job's time window $[\mathrm{est}_j, \mathrm{lct}_j)$.

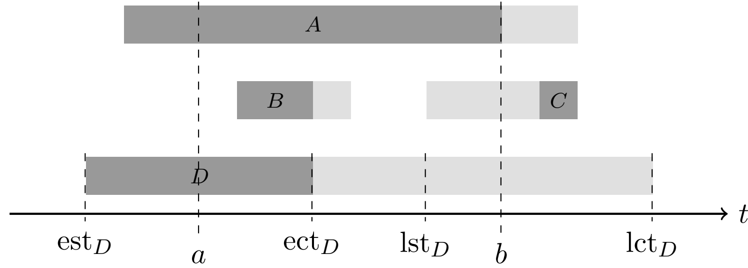


Figure 2.5: Visualization of $e_j^{\mathrm{ER}}(a, b)$. Job A contributes $r_A \cdot (b - a) < r_A \cdot p_A$, job B is fully contained and contributes $r_B \cdot p_B$, whereas job C has zero energy in interval $[a, b)$ and job D intersects with the interval from both sides, hence, $e_D(a, b) = r_D \cdot \min\{\mathrm{ect}_D - a, b - \mathrm{lst}_D\} = r_D \cdot (b - \mathrm{lst}_D)$.

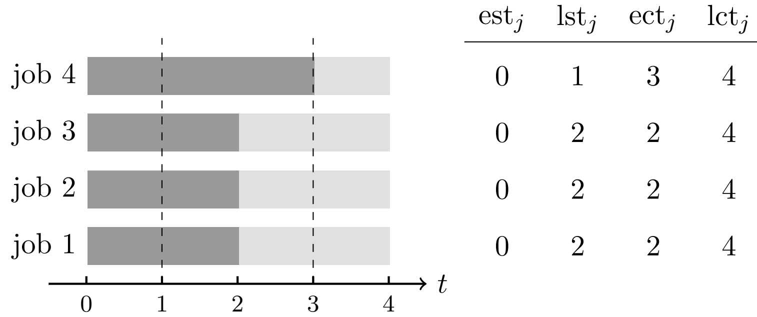
Next, we present the standard energetic checking and propagation rules, see e.g., Baptiste et.al. [22]. Consider a problem setting consisting of n jobs with some lower and upper bounds on the start time variables. If an interval is *overloaded*, i.e., the required energy $E_{\mathcal{J}}(a, b) := \sum_{j \in \mathcal{J}} e_j^{\mathrm{ER}}(a, b)$ is larger than $R \cdot (b - a)$, then the problem is infeasible.

Lemma 2.8. *If for a non-empty interval $[a, b)$, $E_{\mathcal{J}}(a, b) > R \cdot (b - a)$ holds, then the problem is infeasible.*

Algorithm 3: General framework of the propagation algorithms energetic reasoning and edge-finding.

Input: Resource capacity R, set $\mathcal{J}$ of jobs, set of intervals O.
Output: Earliest start times est'_j for each job j or an infeasibility is detected.

1 **foreach** *interval* $[a, b) \in O$ **do**
2   **if** $E_{\mathcal{J}}(a, b) > R \cdot (b - a)$ **then**
3   **return** infeasible.
4   **foreach** *job* j **do**
5   Compute update of earliest start time if overload is created:
6         $\text{est}'_j = \max\left\{\text{est}_j, a + \left\lceil \frac{1}{r_j}\left(E_{\mathcal{J}\setminus\{j\}}(a, b) - (b - a) \cdot (R - r_j)\right)\right\rceil\right\}$.



	est_j	lst_j	ect_j	lct_j
job 4	0	1	3	4
job 3	0	2	2	4
job 2	0	2	2	4
job 1	0	2	2	4

Figure 2.6: Problem setup of Example 2.3.

Variable bounds are updated by checking whether scheduling a job at its earliest start time would induce an overloaded interval as stated in the following Lemma.

Lemma 2.9. *If for an interval* $[a, b)$ *the conditions* $E_{\mathcal{J}}(a, b) \leq R \cdot (b - a)$ *and* $E_{\mathcal{J}\setminus\{j\}}(a, b) + r_j \cdot (\min\{\text{est}_j + p_j, b\} - \max\{a, \text{lst}_j\}) > R \cdot (b - a)$ *hold, then the earliest start time of job* j *can be updated to:*

$$\text{est}'_j = a + \left\lceil \frac{1}{r_j}\left(E_{\mathcal{J}\setminus\{j\}}(a, b) - (b - a) \cdot (R - r_j)\right)\right\rceil. \tag{2.3}$$

Proofs are given in [22]. The fastest known implementation for energetic reasoning runs in $O(n^3)$ by checking for each job all $O(n^2)$ intervals of interest. See Section 2.1.4 for further information and our study on an approximative criterion for this algorithm to detect overloads of intervals. Algorithm 3 sketches the main steps of energetic reasoning. The algorithm checks in line 2 for each interval whether the problem is already infeasible according to Lemma 2.8. Then, each job is scheduled as early as possible and thereby a job is possibly postponed in line 6 by applying Lemma 2.9.

Example 2.3. Consider a cumulative resource of capacity 2 and four jobs each with a resource demand of 1, an earliest start time of 0 and a latest completion time of 4. Three of these jobs have a processing time of 2. The fourth job has a processing time of 3 instead. Figure 2.6 illustrates this setup. The available energy interval $[1, 3)$ is $(3 - 1) \cdot 2 = 4$. The first three jobs contribute one unit each, whereas the fourth job

adds two units to the required energy. This sums up to $E(1,3) = 5$. This shows that these jobs cannot be scheduled.

Edge-finding Edge-finding can be seen as a special variant of energetic reasoning. In that variant the energy requirement of a job is only considered if the job lies completely in the interval $[a,b)$. It is defined by $e_j^{\mathrm{EF}} := p_j \cdot r_j$ if $\mathrm{est}_j \geq a$ and $\mathrm{lct}_j \leq b$ hold, and $e_j^{\mathrm{EF}} := 0$ otherwise. This clearly leads to weaker bound updates, but can be executed in $O(n^2 \log(n))$ using sophisticated data structures, namely the theta-lambda-trees, see Vilím [262]. All update formulas and infeasibility statements from energetic reasoning are transferred to edge finding by exchanging $e_j^{\mathrm{ER}}(a,b)$ with $e_j^{\mathrm{EF}}(a,b)$ in the corresponding formulas.

The running time of the propagation algorithm has further been improved to $O(kn\log(n))$ by Vilím [261], where the implementation details are tricky. Here, in each execution round only a job with largest update is searched for and therefore the algorithm may need more rounds of execution to reach a fix point. Nevertheless, in many runs, no updates are found and in total this turned out to be a very efficient implementation. Several of these details and suggested improvements can be found by Scotts [233].

Time-table edge-finding Recently, Vilím [263] proposed a lazy edge-finding procedure that incorporates energy requirements from time-tabling into the edge-finding algorithm. It runs in $O(n^2)$ and turned out to be very good in deriving lower bounds. In parallel to our work, this algorithms has also been implemented by Schutt et al. [227] and they report improvements of their lazy clause generation approach that is able to derive better dual bounds for some instances from PSPLib.

Impact of propagation on the solving process Table 2.1 shows the ambiguity of using these different propagation algorithms. It can be observed that highly cumulative instances need the propagations inferred by energetic algorithms, while most instances are solved most efficiently when using the time-tabling propagator. Table 2.1 summarizes these results on the chosen test sets.

Recall the description of Figures and Tables from Section 1.5.3. Column 'nopt' indicates the number of optimally solved instances, 'bprimal' ('bdual') gives the number of instances where the solver found the best primal (dual) bound among the competing solvers and column 'gap' reports the average gap at the end of the solving process or when the time limit has been hit. The last two columns, show the number of instances that are solved by all settings to optimality and columns 'avtime' ('avnodes') represent the average running time (number of nodes) in the arithmetic mean to solve these instances.

On instances from PSPLib (setS and setL), the pure time-tabling algorithm, denoted by 'tt', is able to solve more instances to optimality than all combinations of time-tabling with any of the energetic propagation algorithms, there are edge-finding 'ef', time-table edge-finding 'ttef' and energetic reasoning 'er'. On the unsolved instances, the final gap is smaller on average.

Considering only the subset of the instances that have been solved to optimality by all solvers, the running time increases by a factor between two (for 'ef') and 50 (for 'er'). The merits of the energetic propagators lie in the reduced number of nodes that need to

be explored. There are about 20% less nodes needed in ‘er’ on **setS** and less than half the number of nodes on the set setL.

For the Pack instances, the reduction in the number of nodes is up to a factor of 6 when using energetic reasoning. In case of edge-finding the number of nodes even increases. We believe that this is due to worse branching decisions mainly on the instances that create a large search tree. On several of these instances, the time-table edge-finding algorithm yields the best results by decreasing the number of nodes and solving several instances faster than the pure time-tabling approach. Though on average the running time gets higher. In total from the Pack set, four more instances can be solved to optimality when using time-table edge-finding instead of running only time-tabling.

Table 2.1: Comparison of the usage of the propagation algorithms time-tabling alone or in combination with either edge-finding, time-table edge-finding or energetic reasoning.

setting	nopt	bprimal	bdual	gap	avtime	avnodes
setS (114 instances)					(allopt: 52 instances)	
tt	**93**	114	114	**1.89**	**3.83**	29600.50
ef	86	107	96	2.52	7.49	28154.71
ttef	84	106	88	2.87	13.65	27433.46
er	52	75	58	6.85	223.57	**24155.06**
setL (119 instance)					(allopt: 64 instances)	
tt	**102**	119	119	**0.93**	**2.07**	5405.28
ef	97	114	113	1.30	3.16	4621.36
ttef	90	107	106	1.73	4.16	3416.38
er	64	81	83	3.84	52.04	**2442.64**
Pack (55 instance)					(allopt: 27 instances)	
tt	35	50	**54**	**2.49**	**1.71**	19691.85
ef	30	45	53	2.80	10.24	97730.37
ttef	**39**	**54**	51	2.46	2.24	11256.11
er	33	48	48	3.10	5.45	**3208.63**

2.1.4 An approximative criterion for energetic reasoning

In this section, we concentrate on the evaluation of the energetic reasoning algorithm. Its merit lies in a much stronger domain propagation rule by which the number of search nodes can be noticeably reduced. Infeasible nodes are detected much earlier than by time-tabling or edge-finding. The adjustment of lower and upper bounds on the start time variables is quite elaborate by using volume arguments. However, the running time of this algorithm is cubic in the number of jobs which often slows down the whole solving process, see e.g. Table 2.1.

Initial experiments to improve the running times when using this algorithm have been carried out. In these experiments only sub intervals of restricted length have been used or the algorithm has been executed only every few depths of the search tree. Doing so, we have not been able to come up with meaningful better results. Faster running times usually came together with less instances that could be solved to optimality. We studied what kind of jobs are most often updated by this procedure and how the intervals are related to these updates. It turned out that jobs with high demands are updated most often, whereas the sizes of the intervals range from short to long. Hence, the idea was to estimate promising intervals.

To this end, we derive a necessary condition for energetic reasoning to detect infeasibilities. The condition is based on a *relative energy histogram*. We show that this histogram underestimates the true energy requirement of an interval by a factor of at most $\frac{1}{3}$ and can be computed efficiently in $O(n\log(n))$. We embed this approximative result in a parametrically adjustable propagation algorithm which detects variable bound adjustments and infeasibilities in the same run.

As our computational results reveal, the presented algorithm remarkably reduces the total computation time for solving instances from the PSPLib [162] in contrast to the pure energetic reasoning algorithm. A decrease in the running time by a factor of 4 is observed on instances from PSPLib if our approximative criterion is used with an approximation factor of about 1.0. Hence, our estimator for relevant intervals to be checked is well-suited on these instances. In contrast, on the highly cumulative Pack instances, it is best to use the common implementation of energetic reasoning.

We point out that these improvements do not yet make this algorithm competitive with the standard time-tabling algorithm. Hence, all experiments carried out for this study are obtained by using energetic reasoning as the only propagation algorithm of the cumulative constraint.

Outline We start by quoting related work in more detail than done in Section 2.1.3.2. Then, we derive a necessary condition for energetic reasoning to be successful and embed it into the standard energetic reasoning procedure. Finally, experimental results show that applying the criterion to all intervals with a factor of one yields the best running times on the instances from PSPLib.

2.1.4.1 Related work

Baptiste et.al. [22] provide a detailed overview on the main constraint programming techniques for cumulative scheduling. Therein, several theoretical properties of energetic reasoning are proven. A more general idea of *interval capacity consistency tests* is given by Dorndorf et.al. [95]. In the same paper, unit-size intervals are considered as a special

case, which leads to the time-tabling algorithm [159]. Recently, Kooli et.al. [165] used integer programming techniques in order to improve the energetic reasoning algorithm. This approach extends the method presented by Hidri et.al. [141], where the parallel machine scheduling problem has been considered. In both works only infeasibility of a subproblem is checked; variable bound adjustments are not performed.

In order to detect infeasibility, $O(n^2)$ time-intervals need to be considered [22]. These intervals correspond to the start and completion times of the jobs and are precisely determined by the following sets:

$$\begin{aligned} O_1 &:= \bigcup_j \big(\{\mathrm{est}_j\} \cup \{\mathrm{ect}_j\} \cup \{\mathrm{lst}_j\}\big), \\ O_2 &:= \bigcup_j \big(\{\mathrm{lct}_j\} \cup \{\mathrm{ect}_j\} \cup \{\mathrm{lst}_j\}\big) \text{ and} \\ O(t) &:= \bigcup_j \{\mathrm{est}_j + \mathrm{lct}_j - t\}. \end{aligned}$$

The relevant intervals to be checked for energetic tests are given by $(a, b) \in O_1 \times O_2$, for a fixed $a \in O_1 : (a, b) \in O_1 \times O(a)$, and for a fixed $b \in O_2 : (a, b) \in O(b) \times O_2$, with $a < b$. These are $O(n^2)$ such intervals. Note that an interval $[\mathrm{lst}_j, \mathrm{ect}_j)$ corresponds to the core of job j, hence, deductions made by energetic reasoning include those of the time-tabling algorithm. In case of feasibility tests, we are able to restrict the set of intervals that need to be considered. Whether such restrictions can also be made for variable bound adjustments is an open problem.

2.1.4.2 Restricted energetic reasoning

Recall the definition of the required energy of some job j in interval $[a, b)$ for energetic reasoning from (2.2) :

$$e_j(a, b) := \max\{0, \min\{b - a, p_j, \mathrm{ect}_j - a, b - \mathrm{lst}_j\}\} \cdot r_j.$$

Energetic reasoning compares the available energy to the requested energy for certain intervals. Therefore, it is more likely to detect variable bound adjustments if the bounds are tight, i.e., if the domain $[\mathrm{est}_j, \mathrm{lst}_j]$ of job j is small. If the bounds are loose and small intervals are considered, a job may contribute almost no energy to that interval or in case of large intervals not enough energy is required in order to derive any adjustments. This is a clear drawback as we are faced with a very time-consuming algorithm. To this end, we identify intervals that are promising to detect infeasibilities and variable bound adjustments.

Estimation of relevant intervals Let us consider one resource with capacity R and cumulative demands r_j for each job j. The total energy requirement of job j is given by $e_j = p_j \cdot r_j$. We measure the *relative energy consumption* by $\tilde{e}_j := \frac{e_j}{\mathrm{lct}_j - \mathrm{est}_j}$.

Furthermore, we define the *relative energy histogram* $\tilde{E} : \mathbb{N} \to \mathbb{R}$ and the *relative energy* $\tilde{E}_{\mathcal{J}}(a, b)$ of an interval $[a, b)$ for a set of jobs $\mathcal{J}$ by:

$$\tilde{E}_{\mathcal{J}}(t) := \sum_{j \in \mathcal{J} : \mathrm{est}_j \le t < \mathrm{lct}_j} \frac{e_j}{\mathrm{lct}_j - \mathrm{est}_j} \qquad \text{and} \qquad \tilde{E}_{\mathcal{J}}(a, b) := \sum_{t=a}^{b-1} \tilde{E}(t).$$

This histogram approximates the required energy $E(a,b)$ computed by energetic reasoning for each point in time, as we prove in Theorem 2.10.

Theorem 2.10. *Let an arbitrary non-empty interval* $[a,b)$ *be given. Then*

$$\alpha \cdot E(a,b) \leq \tilde{E}(a,b)$$

with $\alpha > \frac{1}{3}$.

Proof. Let $[a,b)$ be a non-empty interval. Intuitively, we denote by $\tilde{e}_j(a,b)$ the relative energy that job j contributes to interval $[a,b)$. Hence,

$$\tilde{e}_j(a,b) = \frac{p_j \cdot r_j}{\mathrm{lct}_j - \mathrm{est}_j} \cdot (\min\{\mathrm{lct}_j, b\} - \max\{\mathrm{est}_j, a\}). \tag{2.4}$$

We show the approximation factor α for each job separately. By linearity of summation, the theorem follows. First, we show that we can restrict the study to the case where $\mathrm{est}_j \leq a < b \leq \mathrm{lct}_j$.

If the energy is underestimated in $[a,b)$, then (2.4) yields $\mathrm{est}_j < a$ or $\mathrm{lct}_j > b$, since otherwise $\tilde{e}_j(a,b) = e_j(a,b)$. Assume $\mathrm{est}_j \leq a < \mathrm{lct}_j < b$. Then, $e_j(a,b) = e_j(a, \mathrm{lct}_j)$ and $\tilde{e}_j(a,b) = \tilde{e}_j(a, \mathrm{lct}_j)$ holds. Applying a symmetrical argument to $a < \mathrm{est}_j < b \leq \mathrm{lct}_j$, we can restrict the setting to $\mathrm{est}_j \leq a < b \leq \mathrm{lct}_j$.

Now, we consider all possibilities for which the minimum is attained in (2.2).

Case 1. Consider the case $e_j(a,b) = p_j \cdot r_j$ and $p_j \leq b - a$. That means, the job is fully contained in $[a,b)$, i.e., $[\mathrm{est}_j, \mathrm{lct}_j) \subseteq [a,b)$. Hence, $\tilde{e}_j(a,b) = e_j(a,b)$.

Case 2. Assume the following two properties:

(i) $1 \leq \min\{\mathrm{ect}_j - a, b - \mathrm{lst}_j\} < \min\{b - a, p_j\}$, and

(ii) $e_j(a,b) = \min\{\mathrm{ect}_j - a, b - \mathrm{lst}_j\} \cdot r_j$.

Thus, $\alpha' := \tilde{e}_j(a,b)/e_j(a,b)$ yields:

$$\alpha' = \frac{p_j(b-a)}{(\mathrm{lct}_j - \mathrm{est}_j) \cdot \min\{\mathrm{ect}_j - a, b - \mathrm{lst}_j\}} \overset{(i)}{>} \frac{\max\{p_j, b-a\}}{\mathrm{lct}_j - \mathrm{est}_j}.$$

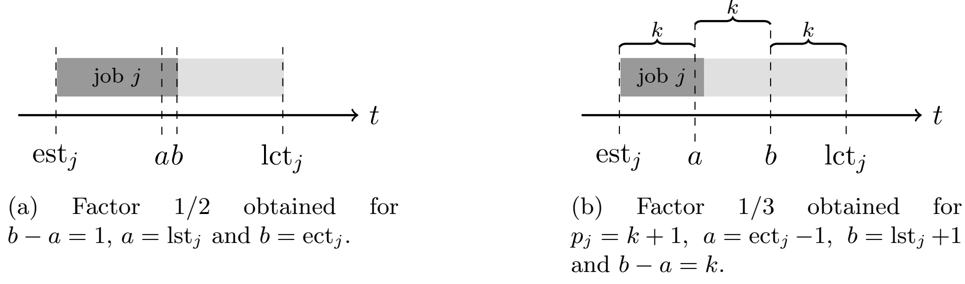
Now, we minimize α' with respect to $1 \leq \min\{\mathrm{ect}_j - a, b - \mathrm{lst}_j\}$. The last condition implies that $\mathrm{est}_j > a - p_j$ and $\mathrm{lct}_j < b + p_j$. The best value for α is given for large intervals $[\mathrm{est}_j, \mathrm{lct}_j)$ obeying this condition. This yields $b - a = k$ and $p_j := k+1$ for some $k \in \mathbb{N}$, such that $\alpha' = \max\{k+1, k\}/(3k) > \frac{1}{3}$.

Case 3. Finally, consider the case where $b - a < \min\{p_j, \mathrm{ect}_j - a, b - \mathrm{lst}_j\}$ and $e_j(a,b) = (b-a) \cdot r_j$ hold. That means, the job is executed at each point in time in $[a,b)$, i.e., $[a,b) \subseteq [\mathrm{lst}_j, \mathrm{ect}_j)$. This yields the condition $\mathrm{ect}_j \geq \mathrm{lst}_j + (b-a)$, which is equivalent to $2\,p_j - (b-a) \geq \mathrm{lct}_j - \mathrm{est}_j$. Thus,

$$\tilde{e}_j(a,b) = \frac{p_j \cdot r_j}{\mathrm{lct}_j - \mathrm{est}_j}(b-a) \geq \frac{p_j}{2\,p_j - (b-a)}(b-a) \cdot r_j = \underbrace{\frac{1}{2 - \frac{b-a}{p_j}}}_{=:\alpha''} e_j(a,b).$$

We obtain $\alpha := \min\{\alpha', \alpha''\} > \frac{1}{3}$. □

The special cases that exhibit the approximation factors 1/2 and 1/3 are depicted in Figure 2.7. The proof shows that an underestimation of $E(a, b)$ occurs with factor 1/2 if the *core* of a job, i.e., $[\text{lst}_j, \text{ect}_j)$, overlaps this interval and with a factor of 1/3 if a job is associated with a large interval $[\text{est}_j, \text{lct}_j)$ and intersects just slightly with $[a, b)$. We observe that the relative histogram may overestimate the required energy. E.g., consider one job j with $\text{est}_j = 0$, $\text{lct}_j = 3$ and processing time $p_j = 1$. Then, $e_j(1, 2) = 0$ but $\tilde{e}_j(1, 2) = r_j/3$.



(a) Factor 1/2 obtained for $b - a = 1$, $a = \text{lst}_j$ and $b = \text{ect}_j$.

(b) Factor 1/3 obtained for $p_j = k + 1$, $a = \text{ect}_j - 1$, $b = \text{lst}_j + 1$ and $b - a = k$.

Figure 2.7: Visualization of the worst case instances that yield the approximation factors 1/2 and 1/3.

The following corollary states a necessary condition to detect infeasibilities by energetic reasoning related to the approximative criterion.

Corollary 2.11. *Energetic reasoning cannot detect any infeasibility, if one of the following conditions holds*

(i) for all $[a, b), a < b$, $\tilde{E}(a, b) \leq \frac{1}{3}(b - a)\, R$, *or*

(ii) for all t: $\tilde{E}(t) \leq \frac{1}{3}\, R$.

The histogram $\tilde{E}$ can be computed in $O(n \log n)$ by first sorting the earliest start times and latest completion times of all jobs and then creating the histogram chronologically from the earliest event to the latest event. Since there are at most $2n$ event points (the start and completion times of the jobs) only $O(n)$ changes in the histogram need to be stored.

2.1.4.3 Restricted energetic reasoning propagation algorithm

We now present a restricted version of energetic reasoning which is based on the results of the previous section. According to Corollary 2.11, only intervals $[a, b)$ containing points in time t with $\tilde{E}(t) > \frac{1}{3}\, R$ need to be checked. Note that the cardinality of this set may still be cubic in the number of jobs. We introduce an approach, in which we only execute the energetic reasoning algorithm on interval $[t_1, t_2)$ if

$$\forall\, t \in [t_1, t_2) : \qquad \tilde{E}(t) > \alpha \cdot R$$

holds. For given $\tilde{E}$, this condition can be checked in $O(n)$. If it holds, we check each pair $(a, b) \in O_1 \times O_2$ with $[a, b) \subseteq [t_1, t_2)$ in order to detect an infeasibility or to find variable bound adjustments.

Algorithm 4: Restricted energetic reasoning (propagation of lower bounds).

Input: Resource capacity R, set $\mathcal{J}$ of jobs with earliest start times est_j, and a scaling factor α. Sets of event points O_1, O_2.

Output: Earliest start times est'_j for each job j or an infeasibility is detected.

```
1  Create relative energy histogram Ẽ.
2  Compute and sort event points {est_j, lct_j}_j and sets O_1 and O_2.
3  foreach job j do
4  |  Set est'_j := est_j.
5  end
6  foreach event point t in increasing order do
7  |  if Ẽ(t) ≤ α · R then
8  |  |  continue.
9  |  end
10 |  t_1 := t.
11 |  Let t_2 be the first event point after t with Ẽ(t_1) ≤ α · R.
12 |  foreach (a, b) ∈ O_1 × O_2 : [a, b) ⊆ [t_1, t_2) do
13 |  |  if E(a, b) > (b − a) · R then
14 |  |  |  return infeasible.
15 |  |  end
16 |  |  foreach job j with [a, b) ∩ [est_j, ect_j) ≠ ∅ do
17 |  |  |  if E(a, b) − e_j(a, b) + e_j^left(a, b) > (b − a) · R then
18 |  |  |  |  V := E(a, b) − e_j(a, b) − (b − a) · (R − r_j).
19 |  |  |  |  est'_j := max{est'_j, a + ⌈V/r_j⌉}.
20 |  |  |  end
21 |  |  |  if est'_j > lst_j then
22 |  |  |  |  return infeasible.
23 |  |  |  end
24 |  |  end
25 |  end
26 |  t := t_2.
27 end
```

The procedure is captured in Algorithm 4. Here only the propagation of lower bounds is shown, upper bound adjustments work analogously.

As mentioned before, the relative energy histogram $\tilde{E}(t)$ can be computed in $O(n \log n)$ and needs $O(n)$ space. The sets O_1 and O_2 also need $O(n)$ space and are sorted in $O(n \log n)$. Loops 6 and 12 together consider at most all $O(n^2)$ intervals $O_1 \times O_2$. Loop 16 runs over at most $O(n)$ jobs. The computed value for $E(a, b)$ in line 13 can be used in the remaining inner loops and all other calculations can be done in constant time, such that we are able to bound the total running time.

Corollary 2.12. *Algorithm 4 can be implemented in $O(n^3)$.*

Asymptotically, it has the same running time as pure energetic reasoning, but the constants are much smaller. Compared to the pure energetic reasoning algorithm we only consider large intervals if the relative energy consumption is huge over a long period. The

savings in running time and further influences on the solving process will be discussed in the following section.

2.1.4.4 Computational results

The only scheduling specific propagation algorithm used is energetic reasoning and its parametric variants, using the necessary condition from Corollary 2.11.

Parameter settings According to Theorem 2.10, it suffices to consider only $\alpha > \frac{1}{3}$. Choosing a value close to $\frac{1}{3}$, however, results in checking the vast majority of the intervals, similar to energetic reasoning. To evaluate the impact of different values of α, we run the algorithm with $\alpha \in \{0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2\}$. For comparison, we further show results for $\alpha = 0.1$ which refers to pure energetic reasoning.

Evaluation of all instances Figure 2.8 shows the number of optimally solved instances per approximation factor X, given by setting"ner-X". Here, $X = 01$ corresponds to $\alpha = 0.1$, while $X = 10$ corresponds to $\alpha = 1.0$. We observe that the outcome of our approximative criterion heavily depends on the problem characteristics, i.e., on the instances from PSPLib, it is best to choose a factor of 0.9 or 1.0, whereas on the Pack instances, it is crucial to check all intervals, hence to use a factor of 0.1. With factors larger than 1.0 much less or even no instances can be solved to optimality.

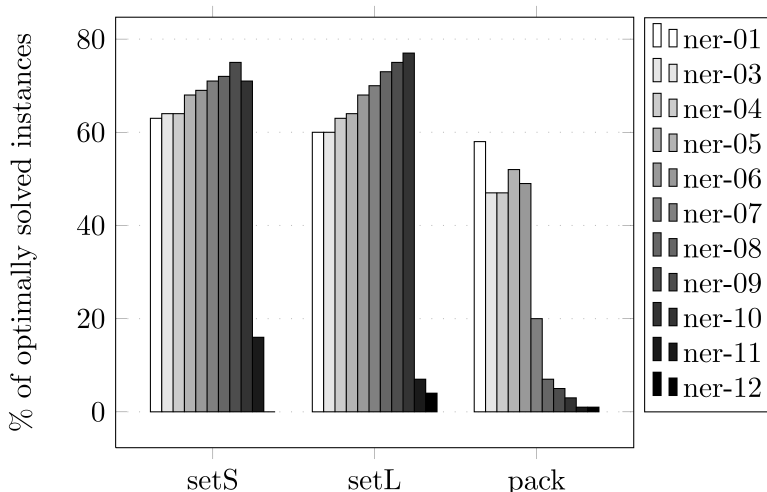


Figure 2.8: Percentage of optimally solved instances per instance set and per setting for the evaluation of the approximative criterion.

For the evaluation of running times and the number of solving nodes, we restrict the experiments to more suitable subsets. We omit the factors 1.1 and 1.2 as always less than 20% of the instances from PSPLib can be solved, compared to at least 60% in the third worst setting. On the Pack instances, we also omit all factors larger or equal to 0.6 as the number of solved instances is too small otherwise.

Now, we turn to the remaining settings in Table 2.2 and compare those instances that are solved to optimality by all settings. On instances from PSPLib, we observe an average speed-up factor of four when using a factor of 0.9 or 1.0 compared to the pure energetic

Table 2.2: Comparison of all optimally solved instances by different factors using the approximative criterion for energetic reasoning.

setting	nopt	bprimal	bdual	gap	avtime	avnodes
setS (114 instances)					(allopt: 72 instances)	
ner-01	72	101	78	4.75	164.89	**7676.36**
ner-03	73	102	78	4.65	173.87	7917.64
ner-04	74	103	80	4.38	144.71	8472.56
ner-05	78	106	83	4.05	101.15	8419.11
ner-06	79	107	87	3.74	69.22	8008.49
ner-07	82	110	93	3.30	46.92	8634.17
ner-08	83	111	101	3.13	39.28	10960.79
ner-09	**86**	**114**	**113**	**2.77**	**31.28**	14002.69
ner-10	81	109	103	3.24	40.25	22888.69
setL (119 instance)					(allopt: 69 instances)	
ner-01	72	98	92	3.15	80.38	1545.84
ner-03	72	98	92	3.14	81.39	1544.01
ner-04	75	101	95	2.94	65.12	1541.33
ner-05	77	103	97	2.81	50.83	**1532.20**
ner-06	81	107	99	2.50	41.54	1544.41
ner-07	84	110	105	2.24	26.18	1565.01
ner-08	87	113	113	2.04	20.06	1953.43
ner-09	90	116	**118**	1.89	**13.8**	2649.30
ner-10	**92**	**118**	115	**1.80**	15.44	5270.06
Pack (55 instance)					(allopt: 20 instances)	
ner-01	**32**	53	54	**3.17**	15.34	**4869.35**
ner-03	26	47	55	3.51	39.76	13043.95
ner-04	26	47	55	3.45	36.39	12058.80
ner-05	29	53	49	3.62	18.62	5876.30
ner-06	27	53	44	4.54	**13.16**	5577.55

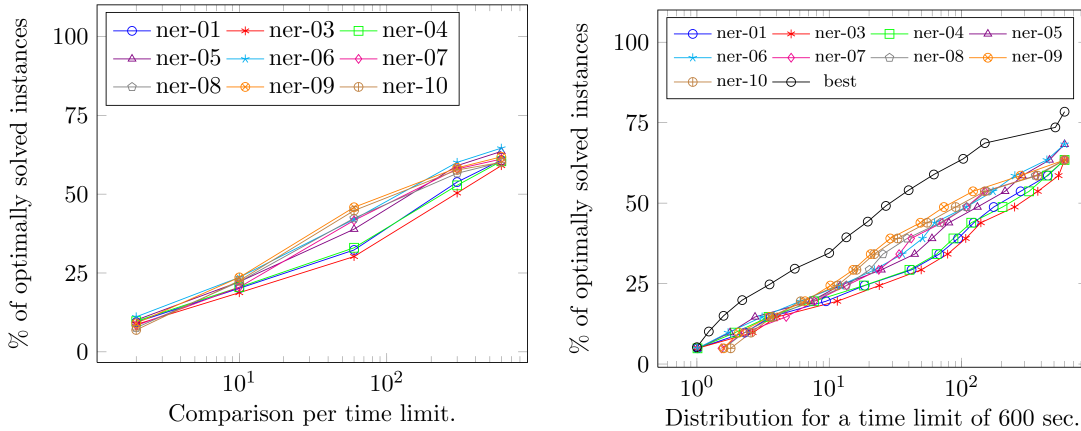


Figure 2.9: Number of optimally solved instances by different strengths of performing energetic reasoning.

reasoning algorithm denoted by ‘ner-01’ though the average number of nodes increases by a factor of up to three. Nevertheless, on the Pack instances, we observe a speed-up factor of two when using a factor of 0.6 or the pure energetic reasoning compared to a factor of 0.3 and 0.4. Here, the number of nodes for factors around 0.3 and 0.4 increases by a factor of two compared to the other settings and hence, also the running times increase. As all other parameters between these settings are absolutely the same, we

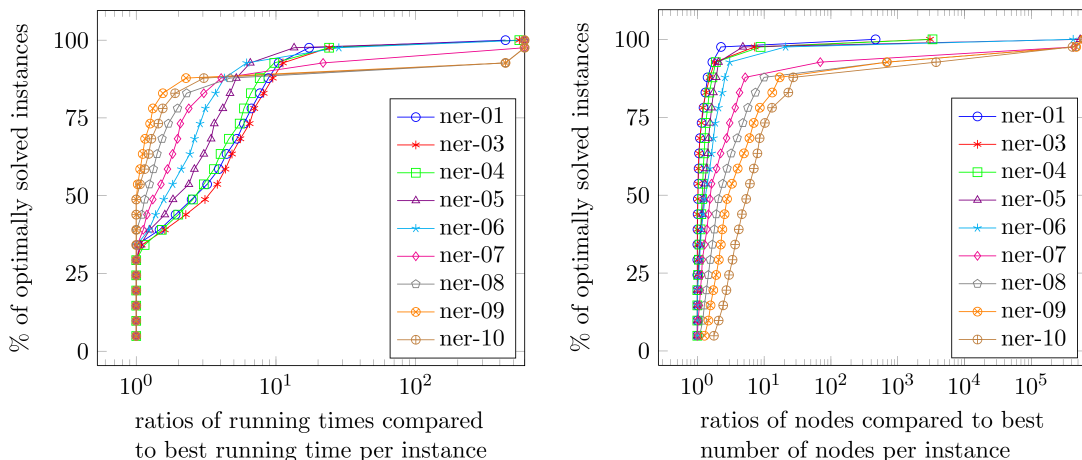


Figure 2.10: Ratios of nodes and running times given for all optimally solved instances from the sets setS, setL and Pack for different strengths of energetic reasoning.

assume that the generic branching rules generate different search trees in which these two settings explore bad directions first. This thesis is supported by the number of best primal solutions. On the corresponding settings, the number of best primal solutions is more than 10% less than by the other settings.

Figures 2.9 and 2.10 emphasize that the lower the approximation factor is chosen, the fewer nodes need to be explored. But considering the running time, it is better to choose a moderate factor of about 0.9 or 1.0 as long as the instances are not highly cumulative.

2.1.4.5 Conclusions

Energetic reasoning is a powerful propagation algorithm but unfortunately, it has a cubic running time which permits its usage throughout branch-and-bound search. We presented a necessary condition for energetic reasoning to detect infeasibilities. Using this condition not all the intervals need to be checked. The condition is based on an average resource demand of all jobs that might be processed at some point in time yielding a relative profile over the planning horizon with at most $2n$ points in time. We embedded this criterion for detecting infeasibilities into the propagation algorithm. For all sub-intervals of interest we check the feasibility of the current subproblem and try to adjust the variable bounds. The relative profile may underestimate the true energy contribution of all jobs by a factor of 1/3. Nevertheless, the computational results reveal that this is a good estimator of relevant intervals as long as the instances are not highly cumulative. The average running time on instances from PSPLib decreases to 1/4 if an estimation factor around 0.9 and 1.0 is used in contrast to checking all possible intervals. Though such a factor is best on instances from PSPLib, this does not carry over to highly cumulative instances (e.g., the Pack instances). On these instances it remains crucial to check all intervals in order to solve the instances efficiently.

Our approach to make the energetic propagation algorithm more efficient is based on determining a subset of promising intervals, hence, to use the propagation rules more efficiently. On the other side, one may also look for stronger arguments in all the different kinds of energetic algorithms. For example, Kooli et.al. [165] propose to solve MIPs in

order to detect intervals with high energy consumption and thereby to detect infeasible nodes early. Within their more elaborate approach it seems to be to expensive to perform variable bound adjustments, hence no results in that direction are reported yet.

2.2 Hybrids of CP and SAT

During a CP search, propagation algorithms are executed in order to tighten the variables bounds. At some stages the problem may become infeasible. The idea from SAT is to find an *explanation* for this infeasibility, which is usually given by the domains of a subset of the variables. Then, similar to a SAT solver, conflict analysis can be applied which enables the solver to use non-chronological backtracking and to learn new clauses or constraints.

In a CP-SAT hybrid solver, propagation algorithms can be performed on the constraints, in particular on those that capture a lot of logical structure and cannot be efficiently encoded otherwise. The most important constraint for which we develop *explanation algorithms* for the corresponding propagation algorithms, is the `cumulative` constraint. The techniques for this constraint are described in Section 2.3. In Section 2.3.1 we analyze the hardness of delivering optimal explanations. It turns out that it is strongly $\mathcal{NP}$-hard to compute minimum-size explanations for bound changes derived from time-tabling propagation, while for energy-based propagation algorithms polynomial-time algorithms are known.

The computational study reveals that by applying conflict analysis, an additional number of instances of about 10% can be solved to optimality within 600 sec. The average number of nodes for the optimally solved instances decreases by 90%, while the computational overhead limits a decrease of the running time to 33% on average. The more efforts are spent to deliver 'good' explanations during conflict analysis, the better for the overall solving process. Further studies per propagation algorithm, on the conflict lengths, or on bound-widening techniques are presented in Section 2.3.3.

We start this section by describing the process of conflict analysis in detail.

2.2.1 Conflict analysis on integer variables

Conflict analysis is an important technique in SAT solvers. When during search a node is encountered to be infeasible, it enables the solver to backtrack to a higher node in the search tree where the infeasibility no longer holds. This procedure is called *non-chronological backtracking*. Additionally, further clauses, called *conflict clauses*, can be derived and added to the global set of clauses.

Conflict analysis on binary variables has been generalized to integer variables and to MIP solving which enables CP and MIP solvers to apply this technique, see Achterberg [1] or Sandholm and Shields [216]. In CP, the concept is used to learn *no-goods*, a selection of variable bounds that lead to infeasibility, see e.g., [120, 213, 244]. In SAT, these are called *conflict clauses*. In our case with integer variables, the goal is to create `bounddisjunction` constraints, i.e., constraints of the form $(\{x_1 \geq 7\} \cup \{x_2 \leq 5\} \cup \{x_3 \geq 8\})$. If no variable assignment satisfies one of these disjunctions, the problem is infeasible. These constraints are used throughout branch-and-bound as domain propagation rules, e.g., if $\{x_1 < 7\} \cap \{x_2 > 5\}$ holds, then $\{x_3 \geq 8\}$ can be immediately propagated. If additionally $\{x_3 < 8\}$ already holds, then an infeasibility is detected. The purpose

of storing and propagating these constraints is to detect an upcoming infeasible state earlier than by performing all or similar propagations and branching decisions again.

During conflict analysis a *conflict graph* is constructed. Special cuts in that graph correspond to the desired conflict clauses. We now describe this procedure in more detail. During a branch-and-bound search, the lower and upper bounds of variables are updated by various propagation algorithms or by branching decisions that generate new subproblems. There are several reasons for a subproblem to become infeasible. E.g., the lower bound of the start time variable of some job is updated to a value larger than the current upper bound; the resource demand of all jobs for some interval is larger than the available capacity; or a relaxation, like the LP relaxation, becomes infeasible. Analyzing such infeasibilities is captured in the notion of *explaining* them.

Recall from Section 2.1.1 that given a propagation algorithm P over a set of variables S together with all bound changes $\mathcal{B}$ performed so far and a constraint $C(S)$ over these variables, we denote the set of feasible bound adjustments by $f_P(S, \mathcal{B}, C(S))$. Observe that for infeasible problems, any bound adjustment is feasible for the state. But there is at least one induced by the applied propagation algorithm that detects the infeasibility.

To explain, bound adjustments or infeasibilities means to analyze the domains of the variables given by $\mathcal{B}$. We introduce this notion formally for lower bound changes, upper bounds are treated equivalently.

Definition 2.13 (Explanation)**.** *Given a propagation algorithm P over a set of variables S, their domains in form of halfspaces $\mathcal{B}$ and a constraint $C(S)$. Let S_j be a variable and* est'_j *be a domain reduction of P, hence $\{S_j \geq \mathrm{est}'_j\} \in f_P(S, \mathcal{B}, C(S))$.*

A set $\mathcal{B}' \subseteq \mathcal{B}$ is an explanation for $\{S_j \geq \mathrm{est}'_j\}$ iff $\{S_j \geq \mathrm{est}'_j\} \in f_P(S, \mathcal{B}', C(S))$.

Intuitively speaking, an *explanation* of an infeasibility or of a bound update is a set of lower and/or upper bounds of variables that, whenever they occur in that combination, lead to an infeasible state or to that bound update. Observe that $\mathcal{B}$ is always an explanation for a domain reduction and an infeasibility but does not yield a reasonable conflict clause.

Starting with an initial explanation for an infeasibility, a conflict graph is constructed. This graph connects each element of the initial explanation via a directed edge with a super sink (symbolizing the infeasibility). Each node in that graph that corresponds to a domain reduction found by some propagation algorithm (besides branching decisions or global bounds) need to be explained, and this explanation is again added to the conflict graph via new nodes and connecting edges.

We remark that for each variable, at most one halfspace out of $\boldsymbol{L}^j$ and at most one halfspace out of $\boldsymbol{U}^j$ is part of an explanation, since if there were more than one lower bound, one would dominate the others. In the conflict graph itself, several bounds of one variable may appear.

When considering propagation algorithms of the `cumulative` constraint we show in Section 2.3.2 that these explanations are in general not unique and bear a huge potential to speed up the solving process. we are able to propagate a clause in the SAT model if all but one literal are false, then the remaining literal must be set to true in order to satisfy the clause. The current assignment of the other variables (being either true or false) of this clause are an explanation for this domain reduction. Hence, the explanation is unique. Already in case of a knapsack constraint, if several items are picked and one huge item does not fit anymore into the knapsack without violating the capacity, there may

be several subsets of the items by which an overload can be detected. For the cumulative constraint, it often becomes hard to compute a set of minimum size, as we show later. We call algorithms that deliver an explanation if multiple explanations may be present, an *explanation algorithm*.

Definition 2.14 (Explanation Algorithm)**.** *An* explanation algorithm *is an algorithm that computes an explanation (a set of halfspaces $\mathcal{B}'$) for a domain reduction from Definition 2.13 under certain objective criteria.*

Some properties are desirable for an explanation algorithm. Since these are carried out frequently, it should have polynomial running time with a low order polynomial though we do not restrict the definition to polynomial time algorithms. The explanations $\mathcal{B}'$ should be of minimum size, since their size correlates to the width of the conflict graph and a small width is expected to create smaller clauses, which are more general and hence more likely to detect bound changes or infeasibilities themselves. Intuitively, the fewer nodes in the graph, the fewer calls to an explanation algorithm are needed. This leads to an objective function in which the total number of bounds is minimized. Similarly, the reported halfspaces should belong to bound changes performed high in the search tree, which makes clauses valid in higher nodes, allows backtracking to a higher node and leads to less calls of explanation algorithms. Hence, we may weight the halfspaces of the explanation with respect to the height in the tree.

Furthermore, since for each variable multiple bound changes may be discovered throughout search, not only the current bounds can be part of the explanation $\mathcal{B}'$, but also earlier bound changes. Imagine that for variable S_j the sequence $L_j^1, L_j^2, .., L_j^k$ of lower bound changes is given. The local lower bound of variable S_j corresponds to L_j^k. In an explanation any bound $L_j^1, ..., L_j^{k-1}$ instead of L_j^k can be contained. Using these bounds for explanations is called *bound-widening*. Then, for each variable multiple alternatives per bound are given. Since bound changes that lie higher in the tree are more desirable, we can weight for each variable the chosen bound differently. Then, we are looking for an optimal explanation with respect to this weighted objective function.

Example 2.4. This example uses propagation algorithms that will be formally introduced later, but are easy to follow with profound knowledge in CP based scheduling algorithms.

The branching decision from Example 2.1 lead to the following deductions according to fully propagated precedence constraints:

- $\{S_B \geq 1\} \implies \{S_C \geq 2\} \implies \{S_D \geq 5\} \implies \{S_E \geq 8\}$,
- $\{S_G \leq 5\} \implies \{S_F \geq 2\}$, and
- $\{S_E \leq 8\} \implies \{S_D \leq 8\} \implies \{S_C \leq 2\} \implies \{S_B \leq 1\}$.

According to the time-tabling algorithm, see Section 2.1.3.2, which could not detect any bound changes before, the halfspace $\{S_A \geq 11\}$ can be deduced which contradicts $\{S_A \leq 8\}$. Hence, the current subproblem after three branching decisions is infeasible.

Different explanations for this update are possible: The cores of the jobs (G, D, E), (C, E) or (B, C, D, E, F, G) lead to possible explanations. A minimum-size explanation for the infeasibility is given by: $\{S_A \leq 8\} \cap \{S_C \geq 2\} \cap \{S_C \leq 2\} \cap \{S_E \geq 8\} \cap \{S_E \leq 8\}$. Where, $\{S_A \leq 8\}$ is a global bound and can be omitted.

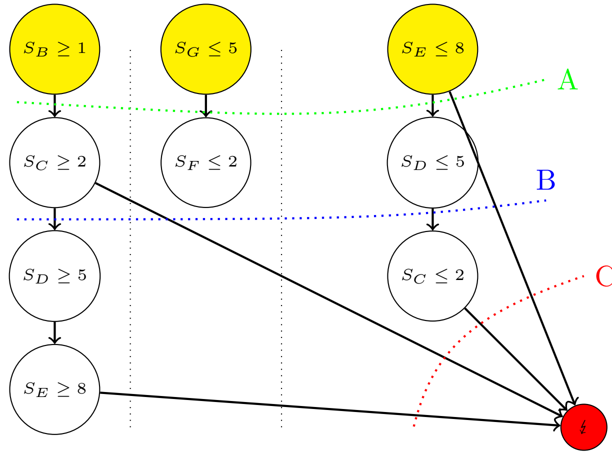


Figure 2.11: A conflict graph for the initial explanation $\{\{S_A \leq 8\} \cap \{S_C \geq 2\} \cap \{S_C \leq 2\} \cap \{S_E \geq 8\} \cap \{S_E \leq 8\}\}$. Yellow vertices are branching decisions, cuts that separate the infeasibility from the frontier of the branching decision are called conflict clauses.

Figure 2.11 shows the resulting conflict graph. All cuts in that graph that separate the branching vertices from the infeasibility $\lightning$ yield conflict clauses. Two cuts, marked by the dotted lines, are of no interest: Conflict clauses induced by the green line ('A') will never occur again, since this combination of branching decisions is pruned. The red line ('C') corresponds to a direct implication of the propagation algorithm and is in general more efficiently encoded in the corresponding propagation routine than by a new bounddisjunction constraint.

The blue line 'B' indicates $(\{S_C \geq 2\} \cap \{S_D \leq 5\} \cap \{S_E \leq 8\}) \implies \lightning$. Since $\{S_E \leq 8\} \implies \{S_D \leq 5\}$, we can learn the bounddisjunction constraint $\{S_C \leq 1\} \cup \{S_E \geq 9\}$ which is the negation of $(\{S_C \geq 2\} \cap \{S_E \leq 8\})$. Due to this constraint in depth level 2, we can immediately add the halfspace $\{S_E \geq 9\}$ because $\{S_C \leq 1\}$ is violated.

Now, we illustrate the concept of bound-widening using the initial explanation consisting of the jobs C and E. We observe that using the global lower bounds for C ($\{S_C \geq 1\}$) and E ($\{S_E \geq 7\}$), the cores are given by $\gamma_C = [2;4[$ and $\gamma_E = [8;10[$, by which time-tabling would also deduce an infeasible halfspace $\{S_A \geq 10\}$. Then, the initial explanation (without global bounds) is $\{S_C \leq 2\} \cap \{S_E \leq 8\}$, which, due to the implication, can be reduced to $\{S_E \leq 8\}$.

Besides creating the conflict graph itself, different methods (*UIP, FUIP, 1-FUIP, All-FUIP*) are known for generating conflict clauses. Some of them are explained next while we refer to [1, 270] for further details. The nodes of the conflict graph can be arranged in depth-levels according to the node in the tree where the bound change happened.

Definition 2.15 (Unique implication point [1])**.** *A* unique implication point (UIP) *of depth level d is a vertex $\ell_d^u \in V$ representing a bound change in depth level d, such that every path from the branching vertex of depth level d to the conflict vertex $\lightning$ goes through $\ell_d^u \in V$ or through a UIP $\ell_{d'}^{u'} \in V$ of higher depth level $d' > d$. The* first unique implication point (FUIP) *of a depth level d is the UIP $\ell_d^u \neq \lightning$ that was fixed last, i.e., that is closest to the conflict vertex $\lightning$.*

The *1-FUIP* scheme only considers the last depth level, whereas *All-FUIP* considers

every single depth level. The FUIP of the first level corresponds to $\{S_D \leq 5\}$ which is necessary to induce the infeasibility besides vertices on higher depth levels and branching decisions. The FUIP of the last level is given by $\{S_E \geq 8\}$. The indicated cut 'A' is the set of branching decisions on a path from the root node and will never be visited again. Cut 'B' is of interest as it induces only three variable bounds, while cut 'C' is directly encoded in the propagation routine that detected this infeasibility.

2.2.2 Explanations for the basic constraints

Recall from Section 2.1.2 the propagation and infeasibility detection algorithms for the precedence and disjunctive constraints. We summarize how the explanations are created w.r.t. each constraint. These explanations are pretty intuitive and there is no room for optimization as there are no choices which bounds to report. They are implicitly given by the lemmata derived in Section 2.1.2.

2.2.2.1 The precedence constraint

The bound changes per precedence constraint can be easily explained:

Lemma 2.16. *Given* $\texttt{precedence}(S_i, S_j, p_i)$*.*
(i) A lower bound change from est_j *to* est'_j *can be explained by* $\{S_i \geq \mathrm{est}'_j - p_i\}$*.*
(ii) An upper bound change from lst_i *to* lst'_i *can be explained by* $\{S_j \leq \mathrm{lst}'_i + p_i\}$*.*

If the problem is infeasible w.r.t. a precedence constraint, then the two jobs must be scheduled in parallel according to their variable lower and upper bounds. This is explained by one lower and one upper bound as condensed in the following lemma.

Lemma 2.17. *Infeasibilities according to Lemma 2.2 are explained by the following set:*

$$\{S_i \geq \mathrm{est}_i\} \cap \{S_j \leq \mathrm{lst}_j\}.$$

2.2.2.2 Explaining the disjunctive constraint

The disjunctive constraint of a disjunctive pair (i, j) can only propagate if one of the jobs must be scheduled locally before or after the other job. The conditions under which Lemma 2.4 can be applied yield directly the explanation.

Lemma 2.18. *Given a disjunctive pair* (i, j)*.*
(i) An update from est_j *to* est'_j *is explained by* $\{S_i \leq \mathrm{lst}_i\} \cap \{S_j \geq \mathrm{est}_j\}$*.*
(ii) An update from lst_i *to* lst'_i *is explained by* $\{S_i \leq \mathrm{lst}_i\} \cap \{S_j \geq \mathrm{est}_j\}$*.*

If the problem is detected to be infeasible due to a disjunctive pair (i, j), then Lemma 2.5 already includes the reason how the variable bounds are related to each other.

Lemma 2.19. *Given a disjunctive pair* (i, j)*. An infeasibility due to this constraint is explained by*

$$\{S_i \leq \mathrm{lst}_i\} \cap \{S_i \geq \mathrm{est}_i\} \cap \{S_j \leq \mathrm{lst}_j\} \cap \{S_j \geq est_j\}.$$

2.3 Explanations for the cumulative constraint

In contrast to the precedence and disjunctive constraint, the propagation algorithms of the cumulative constraint use more elaborate procedures to update variable bounds. Here, we will see that different choices of which bounds or even which jobs are reported play a crucial role for a CP-SAT-hybrid to be efficient.

In Section 2.3.1 we carry out an analytical study on the hardness of explaining e.g., the domain reductions detected by time-tabling and other algorithms. It turns out that for energy-based algorithms such as edge-finding or energetic reasoning, explanations of minimum size can be delivered within reasonable running time $O(n \log(n))$, while explaining updates detected by time-tabling is strongly $\mathcal{NP}$-hard. For this propagation algorithm, we propose an efficient greedy algorithm. The proposed explanation algorithms are presented in Section 2.3.2.

The computational study in Section 2.3.3 shows that using conflict analysis remarkably reduces the average number of nodes in the branch-and-bound tree (by about 90%), while the average running time decreases to 33%. This holds in particular for time-tabling. For the energy-based propagation algorithms, the speed-up factor is only close to two.

2.3.1 Complexity of delivering optimal explanations

In this section we discuss explanation algorithms for the bound changes that are derived by time-tabling, edge-finding and energetic reasoning. Explanation algorithms identify a set of relevant halfspaces in order to explain infeasibilities or bound changes. In case of the cumulative constraint, this is equivalent to identifying a subset of jobs $\Omega \subseteq \mathcal{J}$ that led to the deduction. This holds, because the three propagation algorithms are based on the lower and upper bounds of each job that belongs to the relevant peak in time-tabling or to the interval in energetic reasoning. Since this set of jobs or bounds is in general not unique, we weight the single halfspaces of an explanation differently. Global bounds do not enlarge the conflict graph, hence, we assign a value of zero to the corresponding halfspaces and for all local bounds we assign a value of one. Thus, a job that is part of an explanation will have objective value 0 if both bounds are global, one if exactly one bound is global and two if both bounds are different from the global ones. Furthermore, we will weight the single bounds of a job by the depth level in the tree where they occur. This is done to find out whether it is good to prefer decisions that have been made early during search and whether it is good to use branching decisions in conflicts, as these are at the boundary of the conflict graph and cannot be explained.

2.3.1.1 Problems for the reductions

We start by defining two basic problems and presenting the known complexity results. These two problems will be used in our reductions to show $\mathcal{NP}$-hardness of finding optimal explanations.

Problem: **Minimum Knapsack Covering Problem (MKCP)**

Instance:	A set of n items with weight w_j and cost $c_j \in \mathbb{Z}$ for each item $j = 1, \dots, n$. Positive integers ℓ, W and C.
Cardinality version:	Is there a subset S of ℓ items such that $\sum_{j \in S} w_j \geq W$?
Weighted version:	Is there a subset S of the items such that $\sum_{j \in S} w_j \geq W$ and $\sum_{j \in S} c_j \leq C$?

Though MKCP is equivalent to the Knapsack problem, when looking for an optimal solution, approximability results do not carry over, see [46]. E.g., if in an optimal solution to Knapsack, all items are picked, the equivalent MKCP solution will pick none of the items. An FPTAS works for both, whereas the LP-IP duality gap has ratio two for Knapsack and W for MKCP. We summarize the main hardness results for MKCP.

Proposition 2.20.

(i) The cardinality version of MKCP can be solved in polynomial time.

(ii) For $w_j \in \{1, 2\}$ the weighted MKCP can be solved in $O(n \log(n))$.

(iii) For general weights there exists an FPTAS.

Proof. To show case (i), we sort the items by non-increasing weight w_j, in $O(n \log(n))$. According to the sorting we add the items to S until $\sum_{j \in S} w_j \geq W$ holds. The number of items is minimum since exchanging any element with another one will not increase the weight, due to the sorting.

In the weighted cases (iii), the problem can be equivalently regarded as a maximum Knapsack which is known to be weakly $\mathcal{NP}$-hard and an FPTAS can be obtained by using the results from Ibarra and Kim [146]. For the special case (ii), in which the weights are chosen from $\{1, 2\}$, we first choose jobs in order of non-increasing demands with weight 1 until the capacity is exceeded or no jobs with weight 1 are left. Then, we pick jobs with weight 2 in non-increasing order of their demands until the capacity is exceeded. After that, some of the picked jobs with weight 1 with lowest demand may be eliminated from the set of picked jobs. Last, some (at least two) of the jobs with weight 1 can be exchanged with larger jobs with weight 2 if the demand of the jobs with weight two is larger. This way, we obtain a minimum weight cover with maximum demand. □

Problem: **Minimum Resource Covering Problem (MRCP)**

Instance:	A set of n jobs fixed in an interval $[s_j, t_j)$ with a bandwidth $c_j \in \mathbb{Z}$ and a weight $w_j \in \mathbb{R}$ for $j = 1, \dots, n$. A demand profile $h : [\min_j s_j, \max_j t_j) \to \mathbb{Z}$. Positive integers ℓ, W.
Cardinality version:	Is there a subset S of ℓ jobs such that for every t : $\sum_{j \in S : s_j \leq t < t_j} c_j \geq h(t)$ holds?
Weighted version:	Is there a subset S of the jobs such that for every t : $\sum_{j \in S : s_j \leq t < t_j} c_j \geq h(t)$ and $\sum_{j \in S} w_j \leq W$ holds?

In case that there is only one point in time to be covered, this problem is a minimum knapsack covering problem. For more than one point in time to be covered, Chakaravarthy et al. [52, 51] show that there exists a 4-approximation algorithm based on a primal-dual scheme for instances with arbitrary cost coefficients.

As one can easily verify, the problem can be equivalently reformulated as a packing problem. Then, a maximum number (weight) of jobs must be selected such that at no point in time the capacity is exceeded. A standard IP formulation for this problem is given by:

$$\max\left\{\sum_{i\in\mathcal{J}} w_i x_i \mid \sum_{i:t\in\gamma_i} r_i x_i \leq (\Gamma_{\mathcal{J}}(t) - (R - r_j + 1))^+ \,\forall\, t, x_i \in \{0,1\} \,\forall\, i\right\}.$$

Note that $(a)^+ = \max\{0, a\}$.

The matrix of this IP has the nice structure of demands occurring consecutively. If all demands are equal, the obtained matrix can be scaled to zero-one entries which finally leads to a network matrix. This problem has been studied under the names *bandwidth allocation* [24, 56, 172], *admission control* [206], *temporal knapsack* [26], *multi-commodity demand flow* [55], *unsplittable flow problem* [16, 17, 34, 53, 54], *interval packing* [60], and *resource allocation* [10, 23, 40, 72, 101, 206].

For a long time the complexity status of the cardinality problem has been open, until Darmann et al. [72] showed weak $\mathcal{NP}$-hardness by a reduction from Partition in 2010. Later, Chrobak et al. [60] prove strong $\mathcal{NP}$-hardness by a reduction from Vertex Cover for a case where demands and weights are equal and both values can get arbitrarily large. The same hardness result holds if all demands are from $\{1, 2, 3\}$ and the capacity is a constant, see [34]. In this reduction, only the weights become arbitrarily large. To this end, Bonsma et al. [34] give a constant factor approximation algorithm, but the existence of a PTAS and whether the problem becomes harder if the processing time of the updated job is larger than one remain open.

According to these results, we summarize for MRCP:

Proposition 2.21.

(i) The cardinality version of MRCP is strongly $\mathcal{NP}$-complete, see [60].

(ii) The weighted version of MRCP is strongly $\mathcal{NP}$-complete even if $w_j = c_j$ for all j, see [60].

(iii) The weighted version of MRCP is strongly $\mathcal{NP}$-complete even if $c_j \in \{1, 2, 3\}$, see [34].

We point out that the cases where there exist only a constant number of different weights and where the cost have bounded ratio remain unattended by the known complexity results. Furthermore, for the cardinality version of MRCP, strong $\mathcal{NP}$-completeness holds for arbitrary large demands whereas weak $\mathcal{NP}$-completeness holds if the cores omit some 'proper' interval structure, see [72], a case which we can usually not assume. The problems MKCP and MRCP will be used in the next section to prove complexity results of the explanation algorithms.

2.3.1.2 Time-tabling – complexity of optimal explanations

In order to perform conflict analysis, we must explain infeasible states and bound changes by sets of variable bounds that induced an update or infeasibility encountered by the propagation algorithm. As remarked before, in an explanation for propagation algorithms

of the cumulative constraint, we are looking for a set $\Omega \subseteq \mathcal{J}$ or equivalently a set of halfspaces $\mathcal{B}'$ (Definition 2.13) that led to the deduction.

When time-tabling detects an infeasible state according to line 3 of Algorithm 2, then the sum of demands of all cores at some point in time t exceeds the capacity R. This can be easily explained.

Lemma 2.22. *An infeasibility due to $\Gamma_{\mathcal{J}}(t) > R$ at a point in time t is explained by a set $\Omega \subseteq \mathcal{J}$ such that*

$$\sum_{j \in \Omega : t \in \gamma_j} r_j > R.$$

Proof. Follows directly from Lemma 2.6 that states that an infeasibility of a peak is determined by the cores of a set of jobs. □

Theorem 2.23. *Given* $\texttt{cumulative}(\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}, R)$ *and an infeasibility due to $\Gamma_{\mathcal{J}}(t) > R$.*

(i) An explanation $\Omega \subseteq \mathcal{J}$ of minimum size for the infeasibility can be computed in $O(n \log(n))$.

(ii) There exists a polynomial time algorithm that computes an explanation $\mathcal{B}'$ of minimum size for the infeasibility running in $O(n \log(n))$.

(iii) Given a weight w_j per job $j \in \mathcal{J}$. There exists an FPTAS that computes an explanation $\Omega \subseteq \mathcal{J}$ of minimum weight $\sum_{j \in \Omega} w_j$ for the infeasibility.

Proof. The explanation of Lemma 2.22 asks for a set of jobs such that their cores overlap point in time t and such that the sum of their demands exceeds the capacity R. This problem is equivalent to MKCP. Hence, Proposition 2.20 applies. □

Observe that the cores can be detected in $O(n)$ and usually only a small subset of the jobs needs to be sorted and reported. Computing minimum-size explanations for bound updates according to line 9 in Algorithm 2 is much more challenging.

Lemma 2.24. *A lower bound update of job j from est_j to est'_j can be explained by the previous lower bound of j and a set $\Omega \subseteq \mathcal{J} \setminus \{j\}$ such that for all intervals $I \in \{[\mathrm{est}'_j - 1, \mathrm{est}'_j)\} \cup \{[a, b) \subseteq [\mathrm{est}_j, \mathrm{est}'_j) \mid b - a = p_j\}$ the following condition holds*

$$\exists\, t \in I : \sum_{i \in \Omega : t \in \gamma_i} r_i > R - r_j. \tag{2.5}$$

Proof. Follows directly from Lemma 2.7 that describes how the bound change is performed. □

According to Lemma 2.24 we need to report a set of jobs at $t = \mathrm{est}'_j - 1$ as the updated job j can be scheduled no earlier than at $t = \mathrm{est}'_j$. Then, for an update from est_j to est'_j each interval in between these two values of size p_j must be covered, as otherwise, the job could be scheduled there. Recall from Figure 2.4 that different peaks may be used in the explanation.

Now, we formulate an integer program to compute an explanation for the bound change of some job j. This IP is used in our experimental study. We assume we are given some weight w_i for each job $i \neq j$. These weights can be chosen according to the

depth level in the search tree, where the bound change has been discovered, or according to the frequency a variable has been involved in former propagations.

$$(IP_{tt}^{p_j}) \qquad \min \sum_{i \in \mathcal{J} \setminus \{j\}} w_i x_i \tag{2.6}$$

$$\sum_{i: t \in \gamma_i} r_i x_i \geq (R - r_j + 1) y_t \qquad \forall\, t \in\ [\mathrm{est}_j, \mathrm{est}'_j) \tag{2.7}$$

$$\sum_{\tau = t - p_j}^{t-1} y_\tau \geq y_t \qquad \forall\, t : \mathrm{est}_j + p_j \leq t \leq \mathrm{est}'_j - 1 \tag{2.8}$$

$$y_t = 1 \qquad t = \mathrm{est}'_j - 1 \tag{2.9}$$

$$x_i \in \{0, 1\} \qquad \forall\, i \in \mathcal{J} \setminus \{j\} \tag{2.10}$$

$$y_t \in \{0, 1\} \qquad \forall\, t \in\ [\mathrm{est}_j, \mathrm{est}'_j). \tag{2.11}$$

Variables y_t denote the decision whether point in time t is reported or not and x_i is one if job i is part of the explanation and zero otherwise. Constraint (2.7) models that at each reported point in time the necessary capacity according to Lemma 2.24 is exceeded by the cores of the reported jobs and constraint (2.8) ensures that at least a peak every p_j time units is reported. Finally, constraint (2.9) guarantees that the point in time before est'_j is reported.

In order to approach the complexity status of this problem, we restrict the problem to the case in which the processing time of the updated job is equal to one ($p_j = 1$).

Theorem 2.25. *Computing an explanation $\Omega \subseteq \mathcal{J}$ of minimum size to explain a bound update detected by time-tabling is strongly $\mathcal{NP}$-hard.*

Proof. In case that the updated job j has processing time $p_j = 1$, the problem corresponds to MRCP and is according to Proposition 2.21 strongly $\mathcal{NP}$-hard. □

In order to use bound-widening techniques, multiple modes that reflect the different combinations of lower and upper bounds are introduced for each job. For this problem we consider the following IP formulation.

$$(IP_{tt}^{w,p_j}) \qquad \min \sum_{i \in \mathcal{J} \setminus \{j\}} \sum_{\ell} w_{i\ell} x_{i\ell} \tag{2.12}$$

$$\sum_{\ell} x_{i\ell} = 1 \qquad \forall\, i \in \mathcal{J} \setminus \{j\} \tag{2.13}$$

$$\sum_{i: t \in \gamma_i} r_i x_{i\ell} \geq (R - r_j + 1) y_t \qquad \forall\, t \in\ [\mathrm{est}_j, \mathrm{est}'_j) \tag{2.14}$$

$$\sum_{\tau = t - p_j}^{t-1} y_\tau \geq y_t \qquad \forall\, t : \mathrm{est}_j + p_j \leq t \leq \mathrm{est}'_j - 1 \tag{2.15}$$

$$y_t = 1 \qquad t = \mathrm{est}'_j - 1 \tag{2.16}$$

$$x_{i\ell} \in \{0, 1\} \qquad \forall\, i \in \mathcal{J} \setminus \{j\}, \ell \tag{2.17}$$

$$y_t \in \{0, 1\} \qquad \forall\, t \in\ [\mathrm{est}_j, \mathrm{est}'_j). \tag{2.18}$$

Here, $x_{i\ell}$ models whether combination ℓ of lower and upper bounds of job i is chosen. All other constraints are similar to the IP formulation without multiple modes. We omit the number of different combinations per job for readability.

2.3.1.3 Energetic reasoning and edge-finding – complexity of optimal explanations

As indicated in Section 2.1.3.2 we use the same formulas for energetic reasoning and edge-finding in order to update variable bounds or in order to detect infeasibilities. We only need to use the correct definition of the energy requirement $e_j(a,b)$ of variable j in interval $[a,b)$. We denote by $e_j(a,b)$ that energy requirement, which can be replaced by $e_j^{ER}(a,b)$ or $e_j^{EF}(a,b)$ for the corresponding propagation algorithm.

Lemma 2.26. *An overload of interval $[a,b)$, with $a<b$, can be explained by a set $\Omega \subseteq \mathcal{J}$ such that*

$$\sum_{j\in\Omega} e_j(a,b) > R\cdot(b-a). \tag{2.19}$$

Lemma 2.27. *A lower bound update of job j to est'_j due to interval $[a,b)$, with $a<b$, intersecting with $[\text{est}_j, \text{ect}_j)$ can be explained by the previous lower bound of job j and a set $\Omega \subseteq \mathcal{J}\setminus\{j\}$ such that*

$$\sum_{i\in\Omega} e_i(a,b) > (R-r_j)(b-a) + (\text{est}'_j - a)\cdot r_j - r_j. \tag{2.20}$$

The lemmata expect an explanation such that a certain threshold is exceeded by the sum of the reported demands. Hence, it is possible to compute such an explanation in polynomial time as stated in the next theorem.

Theorem 2.28. *We consider a cumulative constraint $\texttt{cumulative}(\boldsymbol{S},\boldsymbol{p},\boldsymbol{r},R)$.*

(i) A set $\Omega\subseteq\mathcal{J}$ of minimum size for infeasibilities or bound updates derived by energetic reasoning can be computed in $O(n\log(n))$.

(ii) An explanation of minimum size for infeasibilities or bound updates derived by energetic reasoning can be computed in $O(n\log(n))$.

(iii) There exists an FPTAS to compute an explanation $\Omega\subseteq\mathcal{J}$ of minimum weight for infeasibilities or bound updates derived by energetic reasoning.

Proof. We observe that the problem is equivalent to MKCP and hence the results from Proposition 2.20 apply. □

Summary of complexity results

From the former study we have seen that there are polynomial explanation algorithms that explain infeasibilities detected by the time-tabling algorithm and by energetic arguments as used in edge-finding, time-table edge-finding and energetic reasoning. In contrast, computing explanations of minimum weight becomes weakly $\mathcal{NP}$-hard. Moreover, it is strongly $\mathcal{NP}$-hard to deliver an explanation of minimum size for bound adjustments performed by time-tabling. The results are shown comprised in Table 2.3.

	Time-Tabling (Infeas)	Time-Tabling (Update)	Energetic algorithms
min $\|\Omega\|$	$\mathcal{P}$	strongly $\mathcal{NP}$-hard	$\mathcal{P}$
min $\|\mathcal{B}\|$	$\mathcal{P}$	strongly $\mathcal{NP}$-hard	$\mathcal{P}$
weighted	weakly $\mathcal{NP}$-hard	strongly $\mathcal{NP}$-hard	weakly $\mathcal{NP}$-hard
widening	weakly $\mathcal{NP}$-hard	strongly $\mathcal{NP}$-hard	weakly $\mathcal{NP}$-hard

Table 2.3: Complexity results for explaining propagation algorithms for the `cumulative` constraint.

2.3.2 Explanation algorithms

As we have seen, computing optimal explanations for bound changes derived by the time-tabling algorithm is a challenging research direction, whereas for the energetic propagation algorithms straight-forward procedures exist. Now, we present some heuristics and exact approaches for that task and evaluate them in order to show the effectiveness of minimum size explanations.

Exact explanation algorithms for time-tabling. The first approach to be presented is the direct solving of the MIPs ($IP_{tt}^{p_j}$) or (IP_{tt}^{m,p_j}). In the second approach, we compute a shortest path in an elaborated network of non-polynomial size.

The formulation ($IP_{tt}^{p_j}$) can be immediately used to solve the problem. A drawback of this formulation is that its LP relaxation is quite weak if p_j is larger than one. This can be seen as follows: Consider a lower bound update, to from est_j to $est'_j > \text{est}_j$. The variables (if well-defined) from $y_{est'_j - p_j - 1}$ to $y_{est'_j - 1}$ can take value $1/p_j$, such that the variables $y_{est'_j - 2p_j - 1}$ to $y_{est'_j - p_j - 2}$ can take value $1/p_j^2$, and still constraint (2.8) is satisfied. Hence, in order to strengthen the LP-relaxation we add the following valid inequality:

$$\sum_{t \in I} y_t \geq 1 \qquad \text{for all } I \subseteq [\text{est}_j, est'_j) : |I| = p_j.$$

The advantage of the MIP lies in the ability to include multiple alternatives per job, as they occur in the context of bound-widening. Hence, we use IP_{tt}^{m,p_j} to evaluate the maximum outcome that can be achieved by delivering optimal explanations.

Combinatorially, the problem can be solved via a shortest path computation in an elaborated network of non-polynomial size. Our analysis shows that it is of reasonable size in most cases and can therefore be used instead of solving a MIP. The approach has been introduced by Arkin and Silverberg [10] who considered the maximization variant. We show how to construct a suitable graph in order to also handle the minimization variant for delivering explanations for bound updates where the updated variable has a processing time larger than one.

We define a directed acyclic graph for the weighted variant of MIP (2.6)-(2.11) as follows. For each event point e_t, $t = 0, ..., n$, we define a layer V_t of nodes for all jobs J_t whose core intersects with event point e_t. Only event points $e_t \in \{\text{lst}_i\}_{i \in \mathcal{J} \setminus \{j\}} \cup \{\text{est}_j - 1\}$ need to be considered, i.e., the left bounds of the cores and the latest point in time must be explained. By $E_{t,t+1}$ we denote the set of jobs j such that $e_t \in \gamma_j$ and $e_{t+1} \in \gamma_j$, hence $E_{t,t+1} = J_t \cap J_{t+1}$. Each node $v \in V_t$ corresponds to a set $J_t(v)$ of jobs i with $e_t \in \gamma_i$, such that $\sum_i r_i > R - r_j$. There is an edge between two vertices $u \in V_t$ and $v \in V_{t+1}$ if $J_t(u) \cap E_{t,t+1} \subseteq J_{t+1}(v)$. We associate a distance value d_{uv} with each edge, that is set to $|J_{t+1}(v)| - |J_t(v) \cap E_{t,t+1}|$ in the cardinality case and $\sum_{j \in J_{t+1}(v) \setminus J_t(u)} w_j$ in the

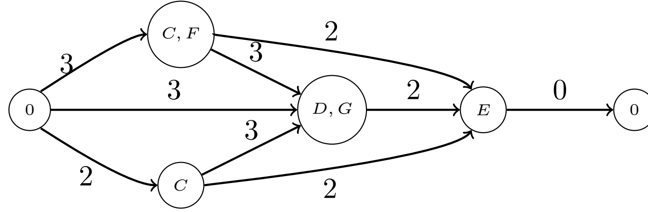


Figure 2.12: A layered graph for the bound update $\{S_A \geq 11\}$. Each additional variable bound adds one to the edge cost.

weighted case. A shortest path in that network corresponds to an optimal solution for a minimum (weight/size) explanation.

In general, there can be exponentially many combinations per layer (exponential in R). But, if the demand r_j of the updated job j is small, we expect only few combinations, since $\sum_i r_i > R - r_j$ and $\sum_i r_i \leq R$ hold. Figure 2.12 illustrates the constructed graph from Example 2.1. E.g., edge $(0, \{C, F\})$ has value 3 as there are three bound changes $\{S_C \geq 2\}, \{S_C \leq 2\}, \{S_F \leq 2\}$ of interest that form the core at event point $e_t = 2$. Observe that $\{S_F \geq 0\}$ is a global bound and can be omitted. The first layer of nodes with $(\{C, F\})$ and $(\{C\})$ corresponds to all combinations of jobs at event point $e_t = 2$ that result in a capacity violation if job A was scheduled in parallel.

Heuristic explanation algorithms for time-tabling. In the following, we describe three different heuristic approaches to derive an explanation for a lower bound updated by time-tabling. These approaches differ in their computational effort. Consider the lower bound update of job j from est_j to est'_j.

Variant 1

Add all variables to Ω whose core intersect with the interval $[\text{est}_j, \text{est}'_j)$.

Variant 2

1. Sort jobs in non-decreasing order w.r.t. their demands.
2. For each $t \in [\text{est}_j, \text{est}'_j)$ with $\Gamma_{\mathcal{J}\setminus\{j\}}(t) > R - r_j$ add jobs $i \in \mathcal{J} \setminus \{j\}$ with $t \in \gamma_i$ to Ω until Condition (2.5) is satisfied.

Variant 3

1. Sort jobs in non-decreasing order w.r.t. their demands.
2. Set $t = \text{est}'_j - 1$.
3. If $t < \text{est}_j$ stop.
4. Explain $\Gamma_{\mathcal{J}\setminus\{j\}}(t)$.
5. Find smallest point in time $t' \in [t - p_j, t)$ such that $\Gamma_{\mathcal{J}\setminus\{j\}}(t') > R - r_j$ holds.
6. Set $t = t'$ and goto (3).

Note that in Variants 2 and 3 we are starting with the largest point in time, i.e., $\text{est}'_j - 1$, and report all cores until we satisfy Condition (2.5). For the remaining peaks we first compute the contribution of previously reported jobs and only add as many new jobs to the explanation until we fulfill Condition (2.5). Variant 1 runs in linear time and

collects all jobs that intersect the interval of interest. Variant 2 explains each peak larger than $R - r_j$ greedily, and Variant 3 tries to report as few peaks as possible. For the last two variants we need $O(n \log n)$ for sorting the jobs in non-decreasing order w.r.t. their demands. Observe that the number of points in time that need to be considered is linear in the number of jobs.

2.3.2.1 Explanation algorithms for energetic reasoning and edge-finding

Recall, that energetic reasoning and edge-finding can be explained similarly. A set $\Omega \subseteq \mathcal{J}$ needs to be computed such that equations (2.19) or (2.20) are satisfied. To construct such a subset of jobs Ω for a lower bound update we compare in our experimental study three different algorithms:

Variant 1
: Add all jobs $i \in \mathcal{J} \setminus \{j\}$ with $e_i(a,b) > 0$ to Ω.

Variant 2
: Add jobs $i \in \mathcal{J} \setminus \{j\}$ with $e_i(a,b) > 0$ to Ω until the Condition (2.20) is satisfied.

Variant 3
: First, sort the jobs with respect to their energies $e_i(a,b)$ in non-increasing order and add jobs to Ω until Condition (2.20) is satisfied.

We assume that the interval $[a,b)$, which inferred the lower bound change, is known a priori. These values can be computed during the execution of the propagation algorithm. Variant 1 corresponds to a simple explanation where all possibly responsible jobs are reported and runs in linear time. Variant 2, which additionally needs a pre-computation of the necessary energy, but still runs in linear time. Because of the sorting step, Variant 3 runs in $O(n \log n)$. Observe that Variant 3 reports a minimum-size explanation with respect to interval $[a,b)$.

Note that in case of an overloaded interval (Lemma 2.26) the above explanation algorithms can be easily adapted by using condition (2.19) as stopping criterion.

2.3.2.2 Time-table edge-finding

In order to explain bound changes derived by energetic propagation algorithms, such as edge-finding, time-table edge-finding or energetic reasoning, we need to identify a set of jobs or variable bounds, that induce enough energy requirement for a certain interval. Observe, though we run a time-efficient algorithm (edge-finding / time-table edge-finding) an execution of energetic reasoning would have deduced the same or even stronger bound updates. Hence, it is possible to explain bound changes by time-table edge-finding or edge-finding by using the energy requirements as given in energetic reasoning.

2.3.3 Computational study

In this computational study we answer the following questions:

- By how much does the use of conflict analysis pay off compared to a pure CP search?

- To which extend should explanation algorithms be carried out?
- How strong is the impact of bound-widening?

We also need to fine-tune the SAT solving capabilities and thereby answer the following questions.

- How many variables to keep in the conflict clauses, and how many conflict clauses shall be kept for propagation?
- To which extend shall the conflict graph be created (1-FUIP, All-FUIP)?

2.3.3.1 Computational environment

For the propagation algorithms time-tabling, edge-finding, and energetic reasoning we have presented explanation algorithms of different strength and different running time. Now, we will compare their ability in order to solve the problems efficiently. As a measure, we consider the number of solved instances and how often the best dual bounds are obtained. Recall, that all algorithms start with the same primal bound, that remains fixed until an optimal solution is found by the destructive search. A destructive search is chosen since SAT techniques work well in that case. On those instances that are solved to optimality by all solvers, we measure the running time and the number of nodes needed. We additionally consider the setting of a pure CP search ("cp") without using conflict analysis in order to measure the outcome of using conflict analysis. For each run we only use the propagation algorithm of interest for retrieving domain reductions due to the cumulative constraints and due to the precedence constraints. All other scheduling specific techniques are disabled.

As mentioned earlier, SCIP has a SAT-like conflict analysis mechanism that builds the conflict graph *backward*, i.e., when an infeasibility occurs, the conflict graph is created on the fly. To avoid an overhead by constructing explanations for bound changes, it is possible to store additional information for each bound change. Since the number of stored bound changes is quite large during the search, the space for such information is restricted to 32 bits each.

In case of energetic reasoning and edge finding we use these bits to store the responsible interval. Otherwise, we would need to search in worst case over $O(n^2)$ interval candidates. Hence, we can use the explanation algorithms stated in the previous section without any additional effort.

2.3.3.2 Computational results

In the following, we use the standard figures and tables as described in Section 1.5.3.

Results for time-tabling

We consider the following proposed variants to explain bound updates and infeasibilities detected by time-tabling compared to a pure CP approach.

- cfttcp: No conflict analysis is applied.
- cftt1: Corresponds to variant 1: each job is reported that intersects the interval $[\text{est}_j, \text{est}'_j)$.

- cftt2: Corresponds to variant 2: greedily each peak is explained.
- cftt3: Variant 3: greedily peak after peak is explained by making steps between the peaks as big as possible.
- cfttbw: Update in single steps and use bound widening to report bounds that are placed higher in the tree

Table 2.4: Comparison of explanation algorithms for time-tabling.

setting	nopt	bprimal	bdual	gap	avtime	avnodes
setS (114 instances)					(allopt: 89 instances)	
cfttcp	93	107	102	1.89	34.23	269041.08
cftt1	91	105	100	1.87	14.63	30857.54
cftt2	95	109	106	1.52	15.27	25957.31
cftt3	97	111	108	1.32	**9.93**	19227.90
cfttbw	**100**	114	114	**1.04**	11.46	**18478.27**
setL (119 instance)					(allopt: 96 instances)	
cfttcp	102	113	117	0.93	29.11	141660.59
cftt1	101	112	113	0.98	19.95	30491.49
cftt2	104	115	117	0.71	10.54	15959.95
cftt3	104	115	117	0.69	10.57	15063.92
cfttbw	**108**	119	119	**0.40**	**7.36**	**9225.35**
Pack (55 instance)					(allopt: 27 instances)	
cfttcp	**35**	54	55	**2.49**	**1.71**	**19691.85**
cftt1	29	48	48	3.24	20.86	73058.85
cftt2	29	48	48	3.16	14.34	53571.00
cftt3	29	48	48	3.19	8.41	52798.26
cfttbw	27	46	48	3.33	10.0	26655.78

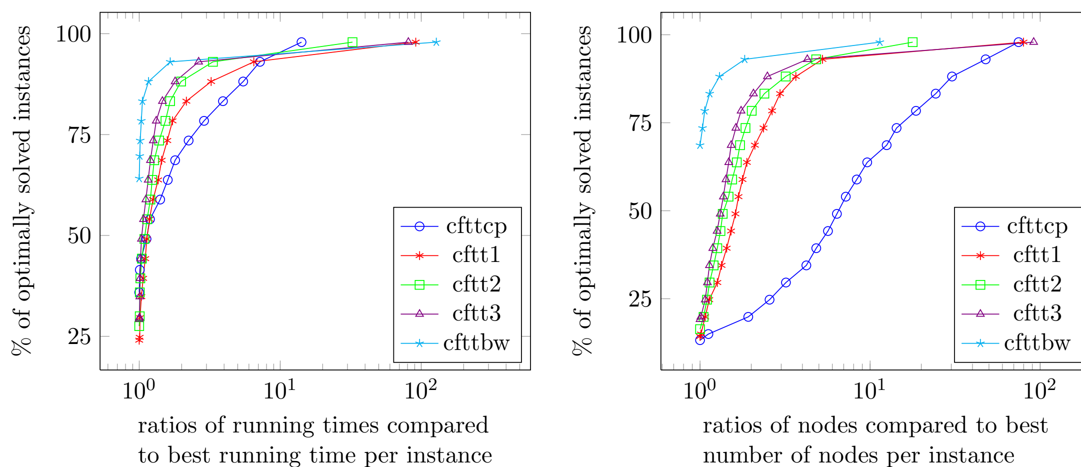


Figure 2.13: Ratios of nodes and running times given for all optimally solved instances from sets setS, setL and Pack for different explanation algorithms for time-tabling.

From Table 2.4 we see that on instance sets 'setS' and 'setL' using more elaborate explanation algorithms pays off in total. The number of instances solved to optimality from set 'setS' increases from 93 instances to 97. This number increases to 100 instances

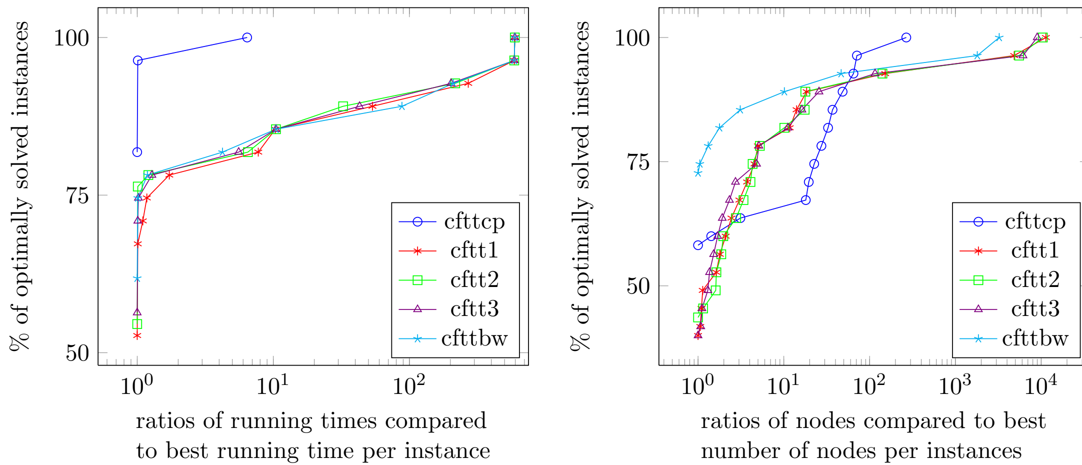


Figure 2.14: Ratios of nodes and running times given for all optimally solved instances from set Pack for different explanation algorithms for time-tabling.

if the bounds are updated step-wise and bound-widening is applied in the explanation algorithms. Similar on instance set 'setL' up to 108 instances can be solved in contrast to 102 in the CP setting. The average running time decreases by more than one half sometimes even to one-third, and on average 90% fewer nodes need to be explored. Compared to the decrease in the number of nodes, the decrease in the running time is low. There are several reasons for this lower decrease in the running time. First, using conflict analysis, branching variables are chosen according to a conflict score and hence, the branching decisions are guided into directions where more bound changes per node can be detected. Hence, the running time per node increases as the propagation algorithms are executed more often. Second, both, constructing the conflict graph and propagating conflict clauses, induce computational overhead. As the reductions in the running times show, this overhead is worth to be made.

On the Pack instances the results reverse. Here, fewer instances can be solved to optimality, 35 in a CP search, with conflict analysis less than 29. The average number of nodes and running time both increase on these instances when using conflict analysis. The running time increases by a factor between 5 and 10 depending on the explanation algorithm used. The reason is that for the highly cumulative instances the conflicts and conflict scores learned are not meaningful enough. Hence, bad branching decisions are made over and over again.

The fact that on instances from PSPLib minimum size explanations work well and on Pack instances do not is once more supported by Figures 2.13 and 2.14. These show that on the Pack instances the CP approach performs well on average while the CP-SAT hybrids are able to solve several instances with fewer nodes and several others needing many more nodes. In contrast, even on average over all instances, the distribution functions concerning running time and number of nodes among those instances that have been solved to optimality in all settings, favor minimum size explanations and bound-widening techniques.

Results for edge-finding

We now evaluate the impact of the presented explanation algorithms when time-tabling (without bound-widening, variant 3) and edge-finding are used. We need to execute the time-tabling algorithm additionally, as without much less instances can be solved to optimality.

- cfefcp: no conflict analysis is applied,
- cfef1: use variant 1 in which all jobs in the considered interval are reported,
- cfef2: use variant 2 in which we stop reporting jobs if enough energy is reported, and
- cfef3: report a minimum size set and use energy requirements as in energetic reasoning.

From Table 2.5 we observe that more instances from instance sets setS and setL can be solved to optimality in about half the running time if conflict analysis is applied. For the Pack instances we experience a reduction in the number of nodes needed to about 1/3, while the running time increases by a factor of almost two and less instances can be solved to optimality if conflict analysis is used.

Table 2.5: Comparison of explanation algorithms for edge-finding.

setting	nopt	bprimal	bdual	gap	avtime	avnodes
setS (114 instances)					(allopt: 85 instances)	
cfefcp	86	106	98	2.52	47.68	152336.56
cfef1	92	112	113	1.86	21.74	17776.33
cfef2	92	112	111	1.88	26.06	20968.84
cfef3	**93**	113	114	1.79	**17.87**	**16740.15**
setL (119 instance)					(allopt: 93 instances)	
cfefcp	97	113	112	1.30	43.45	84151.91
cfef1	**102**	118	117	0.91	**25.02**	**14770.60**
cfef2	**102**	118	117	0.88	28.42	16049.36
cfef3	101	117	118	1.00	27.01	15624.29
Pack (55 instance)					(allopt: 29 instances)	
cfefcp	**30**	55	55	2.80	**18.67**	154822.89
cfef1	29	54	48	3.21	22.01	**54092.50**
cfef2	29	54	49	3.13	22.08	55673.25
cfef3	28	53	49	3.19	29.33	65364.61

Results for time-tabling edge-finding

We consider the following settings. Again, the time-tabling algorithm is additionally used.

- cfttefcp: no conflict analysis is applied,
- cfttef1: all jobs are reported,
- cfttef2: jobs are reported until enough energy is reached, and

- cfttef3: a minimum size set of jobs is reported.

Results on Pack instances in Table 2.6 look similar to edge-finding. It is better to just use domain propagation and not necessarily applying conflict analysis. In particular, now 39 instances from the Pack set can be solved to optimality using a pure CP approach but only 29 or 30 if conflict analysis is applied. For set Pack, it is better to use time-table edge-finding instead of the edge-finding procedure. In contrast to these results, again on instances from 'setS' and 'setL' more instances can be solved to optimality when using conflict analysis while needing 90% fewer nodes and often only half the running time.

Table 2.6: Comparison of different variants to explain time-table edge-finding.

setting	nopt	bprimal	bdual	gap	avtime	avnodes
setS (114 instances)					(allopt: 83 instances)	
cfttefcp	84	109	90	2.87	68.28	118452.31
cfttef1	89	113	112	2.21	29.83	14262.48
cfttef2	89	113	113	2.20	28.51	14034.35
cfttef3	89	113	112	2.20	29.93	13914.35
setL (119 instance)					(allopt: 88 instances)	
cfttefcp	90	108	109	1.73	59.43	60779.76
cfttef1	**100**	**118**	116	1.06	25.48	7721.27
cfttef2	97	115	117	1.25	26.72	7483.07
cfttef3	97	115	116	1.31	28.65	7849.10
Pack (55 instance)					(allopt: 29 instances)	
cfttefcp	**39**	54	**55**	2.46	26.73	157412.52
cfttef1	30	45	49	3.09	30.43	50891.97
cfttef2	29	44	49	3.18	**19.91**	38545.00
cfttef3	30	45	49	3.13	29.01	54677.07

Results for energetic reasoning

Turning to the strongest propagation algorithm, energetic reasoning, that usually suffers from high running times, we can see from the tables that fewer instances can be solved and the average running times increase. We compare the following settings:

- cfercp: no conflict analysis is applied,
- cfer1: all jobs are reported,
- cfer2: jobs are reported until enough energy is reached, and
- cfer3: a minimum size set of jobs is reported.

Here, we see the same picture as for the other propagation algorithms. On instance sets 'setS' and 'setL' more instances can be solved to optimality when using conflict analysis, with a huge decrease in the average number of nodes and needing about half the running time on set 'setS'. On set 'setL' the average running time slightly increases the stronger the explanations are. In contrast, on the Pack instances, fewer instances are solved to optimality and the average number of nodes and running time increase by about a factor of 3.

We show two further comparisons between the instances from PSPLib and the Pack instances that are typical for all the three energy-based propagation algorithms and their

Table 2.7: Comparison of the impact of explanation algorithms for energetic reasoning.

setting	nopt	bprimal	bdual	gap	avtime	avnodes
setS (114 instances)					(allopt: 83 instances)	
cfercp	52	103	74	6.85	217.99	23275.22
cfer1	**62**	112	103	5.62	131.42	4123.25
cfer2	61	111	108	5.64	131.23	4087.04
cfer3	61	111	104	5.69	**127.27**	**4080.80**
setL (119 instance)					(allopt: 88 instances)	
cfercp	64	114	104	3.84	47.96	2430.56
cfer1	**67**	117	113	3.57	**51.85**	**843.39**
cfer2	66	116	107	3.67	55.27	879.18
cfer3	**67**	117	**114**	3.56	53.62	880.63
Pack (55 instance)					(allopt: 24 instances)	
cfercp	**33**	**54**	**55**	**3.10**	**2.97**	**1489.63**
cfer1	24	45	54	3.70	8.39	6737.29
cfer2	26	47	54	3.54	8.82	6688.79
cfer3	26	47	54	3.48	9.07	6638.29

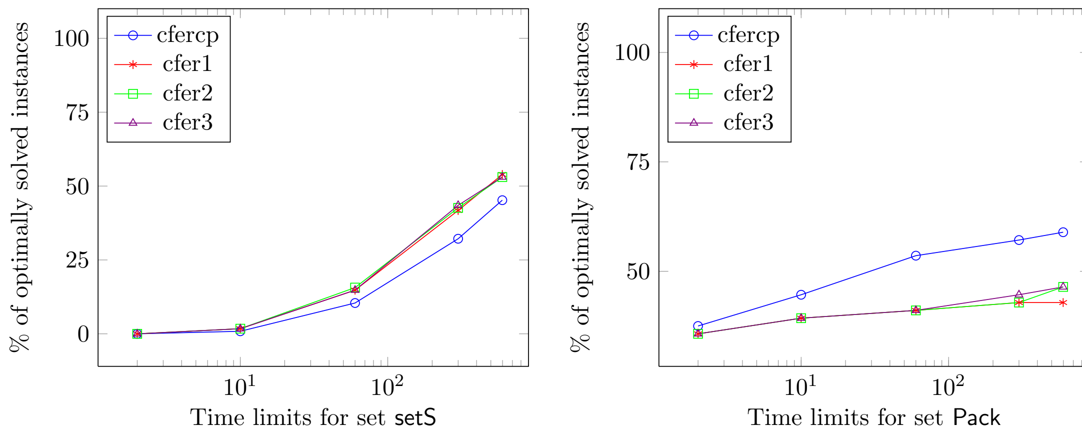


Figure 2.15: Number of instances solved to optimality within time limits of $2, 10, 60, 300$ and 600sec. for different variants to explain energetic reasoning.

explanation variants. Figures 2.15 show the number of instances solved to optimality within different time limits. While on setS, conflict analysis helps to solve more instances as time increases, on the Pack instances, a pure CP approach performs even better over time. Similarly, Figure 2.16 compares the ratios between the number of nodes needed per instance divided by the minimum number of nodes needed in one of the settings. Again, on instances from PSPLib using conflict analysis highly decreases the number of nodes, whereas on the Pack instances, the CP setting performs better with respect to the highest ratio needed. The 20% of the highest ratios are needed by the settings using conflict analysis with a ratio of about 100, in contrast to the highest ratio for the CP approach with a factor of about four.

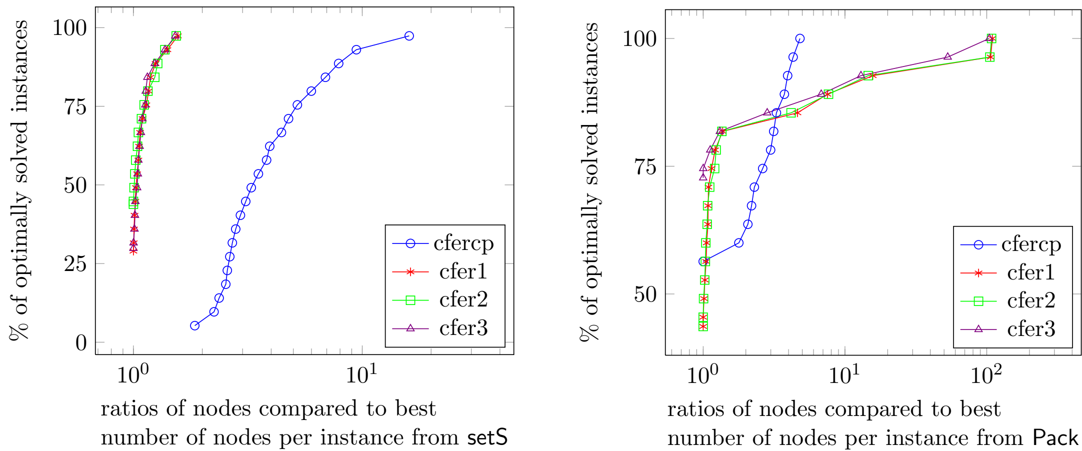


Figure 2.16: Ratio of nodes for all optimally solved instances by different variants to explain enregtic reasoning.

The choice of objective coefficients

In order to deliver good explanations for the time-tabling propagator, we set up two integer programs one with and one without bound-widening. Given that each bound change in SCIP is stored together with the depth level and position in the node where it occurred, we experimented how to best combine bound changes in the explanations. As we will see from the vastly varying results, there are a lot of different choices which bounds to report in a conflict set. In particular if the job whose bound changes has a high demand, in some search states there are several jobs with low demands of which only some need to be reported in order to induce the bound change. In short notes we summarize some choices for the objective coefficients. For each job we need to report a core, hence the lower and upper bound is needed. For each bound, we compute a valuation and obtain a valuation for the job by averaging both valuations. In few settings, we also use the maximum plus the average valuation. If we do so, we state this explicitly. Here are the settings with their objective coefficients.

- cftt4-0: $1 + 2^{-(\text{average percentage of domain compared to global domain})}$
- cftt4-1: 100· (depth + average position)
- cftt4-2: 100· (depth + average position), we punish branching decision by adding +2
- cftt4-3: 2 − (ublocal-lblocal) / (ubglobal-lbglobal)
- cftt4-4: 100· (depth + average position); value of a variable is given by maximum valuation of both bound changes plus the average
- cftt4-5: 100· (depth + average position), +2 if branching decision; val = MAX + average
- cftt4-6: 1 + depth/maxdepth + 0.1 · average position
- cftt4-7: 1 + depth/maxdepth + 0.1 · average position + 0.5 for branching decisions

- cftt4-8: 1000 + depth + average position
- cftt4-9: 1000 + depth + average position; value of a variable is given by maximum valuation of both bound changes plus the average

Now, that MIPs need to be solved, the number of optimally solved instances does not play a role anymore and similarly the average running time is dominated by setting up and solving several MIPs. Hence, we concentrate on the average number of nodes but still give the other numbers for the sake of completeness.

In Table 2.8 we see that the average number of nodes does not vary by more than 10% and considering the different instance sets, there is no clear objective to be preferred. E.g., setting cftt4-4 and cfttt4-7 perform good with respect to that criterion on set setL but cftt4-1 performs best on set setS. We observe from Table 2.8 and Figure 2.17 that the number of nodes needed on set setL is much in favor for the setting cftt4-2 but in contrast this setting is worst on the Pack instances, where cftt4-3 performs best. There seems to be a slight hint that branching decisions should not be used in an explanation, as e.g., cftt4-7 needs between 10 to 20% fewer nodes than cftt4-6. But as this cannot be observed for cftt4-1 and cftt4-2, we cannot experimentally verify this in general.

Table 2.8: Comparison of different objective functions for minimum weight explanations from time-tabling.

setting	nopt	bprimal	bdual	gap	avtime	avnodes
setS (114 instances)					(allopt: 65 instances)	
cftt4-0	71	112	103	4.55	108.45	5623.98
cftt4-1	72	113	96	4.57	107.94	**5356.65**
cftt4-2	71	112	103	4.54	110.63	5548.17
cftt4-3	72	113	104	4.48	107.27	5465.52
cftt4-4	**73**	114	106	**4.41**	**101.7**	5562.25
cftt4-5	71	113	96	4.56	125.45	5574.51
cftt4-6	69	111	94	4.73	126.75	5488.34
cftt4-7	71	112	93	4.72	128.7	5621.80
cftt4-8	67	109	93	4.84	122.21	5637.46
cftt4-9	69	111	95	4.71	118.31	5591.54
setL (119 instance)					(allopt: 73 instances)	
cftt4-0	81	114	111	2.51	41.13	1812.07
cftt4-1	82	115	116	2.43	39.12	1877.26
cftt4-2	74	107	103	3.08	67.9	1732.34
cftt4-3	81	114	112	2.49	36.78	1790.93
cftt4-4	81	114	113	2.49	**33.44**	**1655.41**
cftt4-5	**84**	**117**	114	**2.35**	36.36	1777.78
cftt4-6	83	116	110	2.44	44.69	1930.89
cftt4-7	83	116	113	2.40	39.95	1695.73
cftt4-8	83	116	113	2.42	39.44	1797.84
cftt4-9	82	115	113	2.44	39.03	1827.41
Pack (55 instance)					(allopt: 22 instances)	
cftt4-0	22	55	55	3.87	4.53	165.50
cftt4-1	22	55	54	3.90	3.93	153.64
cftt4-2	22	55	54	3.90	**3.38**	153.55
cftt4-3	22	55	54	3.90	4.94	148.59
cftt4-4	22	55	54	3.90	3.88	154.73
cftt4-5	22	55	54	3.90	4.30	154.32
cftt4-6	22	55	55	3.87	3.42	155.36
cftt4-7	22	55	55	3.87	3.52	**127.14**
cftt4-8	22	55	55	3.87	3.97	153.64
cftt4-9	22	55	55	3.87	4.16	153.09

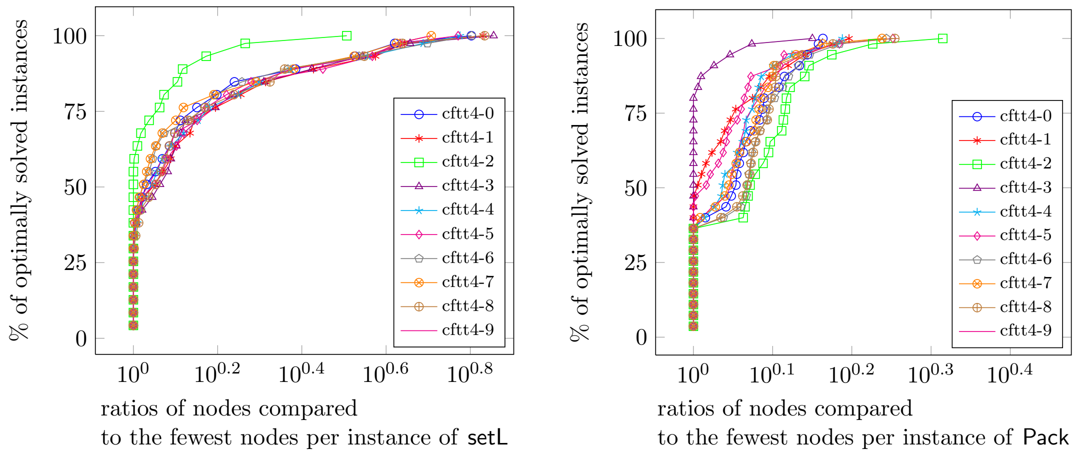


Figure 2.17: Ratios of nodes for different objective coefficients for sets setL and Pack.

Number of variables per conflict

Tuning the conflict analysis tool that is used in SCIP is certainly one of the main components here. We start with an evaluation of the maximum number of variables that can be part of the generated conflict clauses. Recall that the more variables are in a conflict, the more bound changes are needed until such a constraint itself detects a bound change or an infeasibility. Furthermore, the more variables are allowed, the more conflict clauses can be generated which occupy memory and may slow down the solving process. We use the following settings.

- cfX: where X gives the maximum number of variables bounds per conflict, or in terms of SCIP, the number of variable bounds in a `bounddisjunction` constraint. We use $X \in \{0, 6, 8, 10, 12, 14, 16\}$.

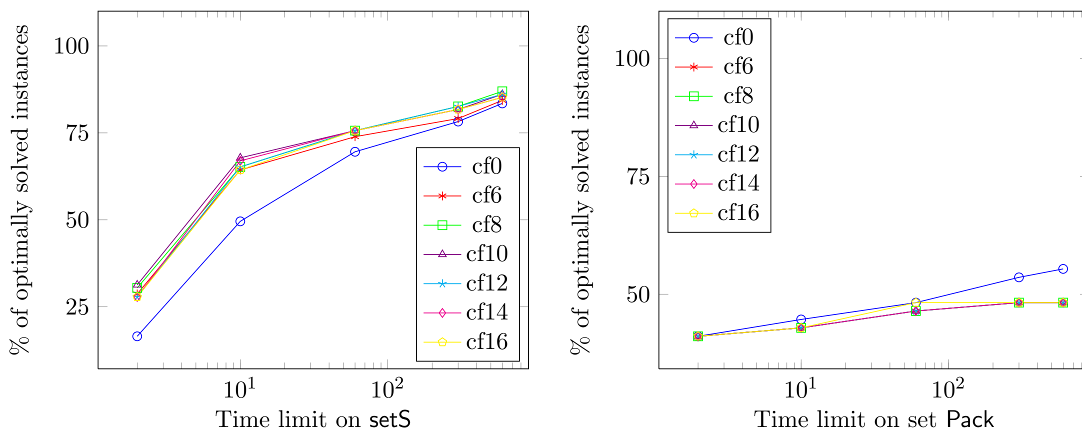


Figure 2.18: Distribution of the percentages of optimally solved instances on sets setS and Pack for time limits 2, 10, 60, 300 and 600 sec.

Table 2.9 shows that on instance sets setS and setL, the more variables are allowed per conflict clause, the fewer nodes are needed. In particular, if no conflict clauses are

Table 2.9: Comparison of results if the maximum number of variables in the conflict clauses is bounded differently.

setting	nopt	bprimal	bdual	gap	avtime	avnodes
setS (114 instances)					(allopt: 94 instances)	
cf0	96	110	101	1.62	42.36	295727.62
cf6	97	111	105	1.31	26.40	44627.70
cf8	**100**	**114**	108	1.03	19.99	34256.85
cf10	99	113	110	1.11	**15.21**	24873.97
cf12	99	113	108	1.13	18.50	25469.24
cf14	98	112	**110**	1.12	18.56	24005.99
cf16	98	112	107	1.23	18.84	**23740.52**
setL (119 instance)					(allopt: 103 instances)	
cf0	104	114	115	0.77	44.53	194749.94
cf6	108	118	116	0.40	12.76	16526.40
cf8	108	118	116	0.40	12.75	16526.40
cf10	108	118	118	0.38	12.99	16563.82
cf12	**109**	**119**	118	**0.33**	14.54	16408.48
cf14	107	117	116	0.48	15.38	15733.69
cf16	108	118	117	0.42	14.40	**15496.35**
Pack (55 instance)					(allopt: 26 instances)	
cf0	**31**	**55**	**55**	**2.67**	**1.88**	16315.08
cf6	27	51	48	3.33	2.69	**10044.23**
cf8	27	51	48	3.36	3.02	10461.85
cf10	27	51	48	3.40	3.86	12373.88
cf12	27	51	48	3.42	4.53	11006.65
cf14	27	51	48	3.42	5.11	12438.38
cf16	27	51	48	3.42	4.76	10573.23

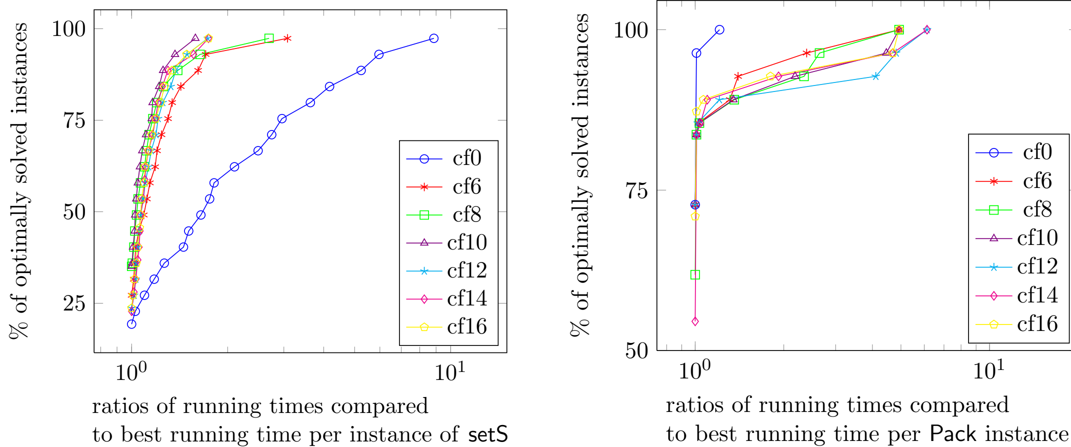


Figure 2.19: Ratios of running times if the maximum number of variables per clause is bounded.

generated (setting cf0), then up to ten times more nodes are needed to solve the same amount of instances to optimality. This is almost similar to using a pure CP approach. We also observe that the running time decreases to a third if conflict clauses are created and propagated. On set setS the best average running time is obtained with an upper bound of 10 variables, while using an upper bound of 8 variables one more instance can be solved to optimality. On set setL, most instances are solved if up to 12 bounds are

part of a conflict clause, whereas the best average running times are obtained for 6 to 10 variables.

Again the results on the Pack instances show a completely different conclusion, see Figures 2.18 and 2.19. Here, it is best in terms of average running time and in the number of solved instances to generate no conflict clause. The slightly lower number of solving nodes when conflict clauses are generated indicates their weakness in contrast to the other two instance sets.

To conclude, on instances from PSPLib it is worthwhile to allow up to between 8 and 10 variable bounds per conflict clause and we chose 10 for our study. On the Pack instances, a pure CP approach currently performs best.

Impact of propagating conflict clauses

The former results already indicate that it does not always pay off to keep conflict clauses for propagation. In particular on the Pack instances this can be observed on each instance.

Some other decisions that need be be made when using conflict analysis and generating conflict clauses is the amount of conflict clauses that are kept. E.g., all conflict clauses that have less than the indicated number of variables or only one clause per depth level can be stored. Conflict analysis can be applied until the first unique implication point of the first depth level is found or all depth levels are considered. This again has an impact on the generated conflict clauses and their capability to propagate. Additionally, if some of the conflict clauses do not propagate they are subject to aging and it is possible to delete them from the search space or to still keep them. If they can be deleted, they are called *dynamic* conflict clauses. Table 2.10 summarizes the settings. The settings are denoted by "cfXYZ". Some combinations can be naturally omitted. If no constraints are stored, we do not need to care whether the conflict clauses are dynamic or not. But it makes a difference if all fuip levels are considered or only one even though no conflict clauses are stored as conflict analysis allows backtracking and collects conflict scores per variable that are updated whenever a variable bound is reported to the conflict graph.

Table 2.10: Settings for conflict analysis denoted by "cfXYZ".

X	maximum number of conflict clauses stored per iteration	Y	FUIP levels	Z	dynamic conflict clauses
0	none			0	no
1	at most one	1	one level	1	yes
-1	all	-1	all levels		

We first elaborate on the usage of dynamic conflict clauses indicated by the third component Z of each setting cfXYZ. Over all instance sets we observe in Table 2.11 that, if the conflict clauses are dynamic, between two and five more instances can be solved to optimality independent of how the other parameters X and Y are set. Though the average number of nodes increase between 10% to 30% (cf110 vs. cf111) if the conflict clauses are dynamic, the average running time decreases between 5% and 20%. Hence, it is better to use dynamic conflict clauses.

If no conflict clauses are generated, then on the set setL it turns out that it is best to perform conflict analysis until the first unique implication point is reached and it

Table 2.11: Comparison of SCIP parameter settings for conflict analysis.

setting	nopt	bprimal	bdual	gap	avtime	avnodes
setS (114 instances)					(allopt: 89 instances)	
cf010	**96**	111	103	1.61	21.47	140735.64
cf0-10	95	110	101	1.74	22.78	143423.56
cf-110	92	107	98	1.86	13.79	16685.79
cf-111	**96**	111	104	**1.48**	**7.43**	22013.74
cf-1-10	92	107	100	1.80	10.51	**13864.30**
cf-1-11	93	108	103	1.81	9.29	18002.62
cf110	91	106	98	1.94	10.49	17787.26
cf111	**96**	111	102	1.54	8.14	25436.54
cf1-10	93	108	101	1.72	12.83	17553.35
cf1-11	94	109	105	1.68	10.80	22003.78
setL (119 instance)					(allopt: 95 instances)	
cf010	**106**	**118**	116	**0.61**	21.31	79250.72
cf0-10	98	110	112	1.32	29.86	112970.42
cf-110	103	115	116	0.81	8.98	11032.54
cf-111	105	117	**117**	0.64	**8.49**	15497.58
cf-1-10	103	115	114	0.77	11.86	8534.00
cf-1-11	103	115	115	0.79	11.46	10923.60
cf110	104	116	115	0.78	12.37	15090.55
cf111	104	116	115	0.72	9.46	18801.40
cf1-10	103	115	113	0.79	14.66	**10913.29**
cf1-11	104	116	114	0.74	11.65	14551.72
Pack (55 instance)					(allopt: 26 instances)	
cf010	29	53	54	2.81	**2.15**	24400.85
cf0-10	**31**	**55**	**55**	**2.68**	3.35	43404.81
cf-110	26	50	48	3.52	4.38	**15528.38**
cf-111	28	52	48	3.18	2.55	16936.42
cf-1-10	28	52	48	3.35	4.77	18656.38
cf-1-11	28	52	48	3.30	8.87	33802.62
cf110	27	51	48	3.48	4.68	17168.69
cf111	29	53	48	3.16	3.20	19275.23
cf1-10	28	52	48	3.40	5.67	19311.88
cf1-11	28	52	48	3.30	3.16	18549.31

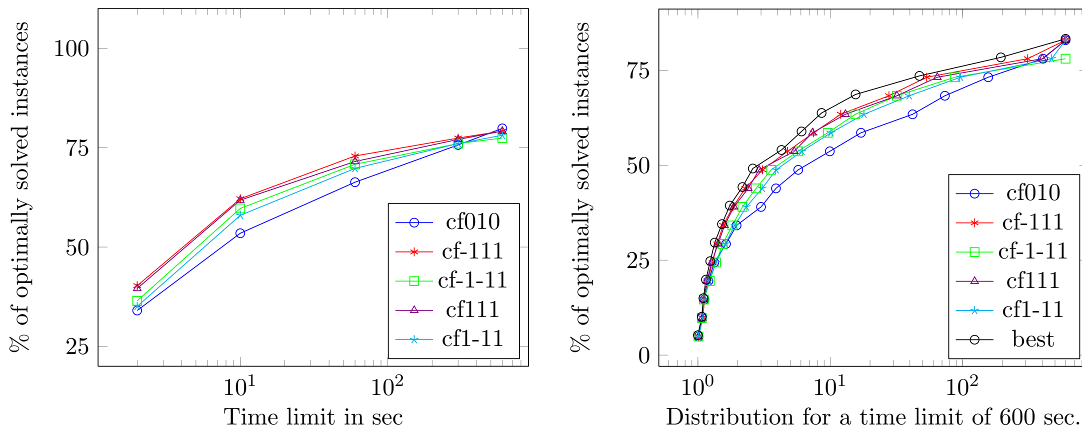


Figure 2.20: Number of optimally solved instances obtained by different parameter settings of conflict analysis.

often does not pay off to collect scores on higher depth levels. In contrast, on the Pack instances, creating the whole conflict graph to higher depth levels pays off, as two more

Table 2.12: Comparison of SCIP parameter settings for conflict analysis.

setting	nopt	bprimal	bdual	gap	avtime	avnodes
setS (114 instances)					(allopt: 91 instances)	
cf010	96	112	104	1.61	27.08	185958.90
cf-111	96	112	105	**1.48**	14.39	33852.65
cf-1-11	93	109	104	1.81	14.20	**23090.45**
cf111	96	112	104	1.54	**11.06**	32357.18
cf1-11	94	110	106	1.68	15.16	27337.46
setL (119 instance)					(allopt: 101 instances)	
cf010	**106**	**118**	116	**0.61**	41.85	169177.14
cf-111	105	117	117	0.64	**14.99**	29867.33
cf-1-11	103	115	115	0.79	17.51	**22987.39**
cf111	104	116	115	0.72	20.60	40055.12
cf1-11	104	116	114	0.74	20.40	29972.90
Pack (55 instance)					(allopt: 28 instances)	
cf010	29	55	55	**2.81**	6.11	105131.46
cf-111	28	54	49	3.18	9.98	80272.14
cf-1-11	28	54	49	3.30	11.45	64788.46
cf111	29	55	49	3.16	6.53	**47973.96**
cf1-11	28	54	49	3.30	**5.89**	50488.29

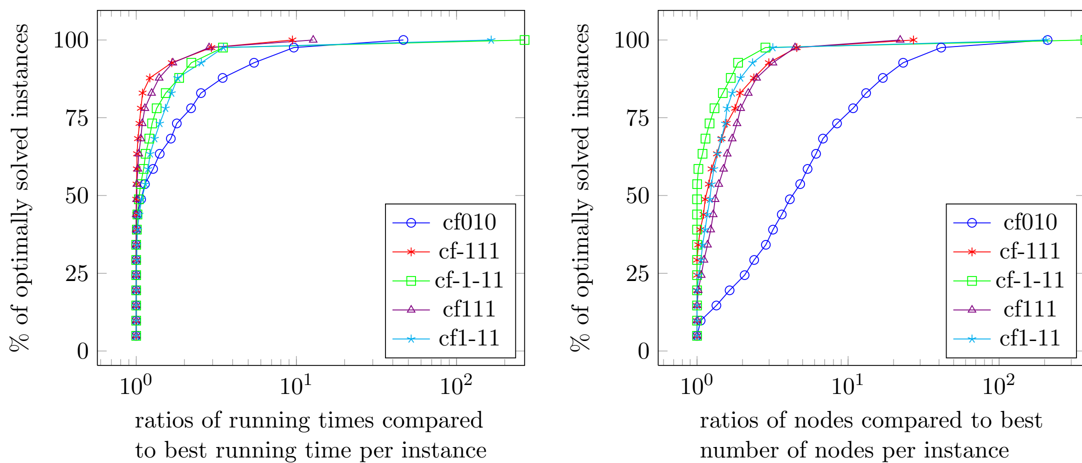


Figure 2.21: Ratios of nodes and running times given for all optimally solved instances from sets setS, setL and Pack using different parameter settings for conflict analysis.

instances can be solved and half the number of nodes is needed. This again reveals that the information gained on these instances is bad and misleading and that's why it is worthwhile on these instances to collect information on the whole path to the root node and not to stop after the fuip is reached.

As seen, it is better to set the bounddisjunction constraints dynamic. Table 2.12 compares more compressed the remaining settings. The most instances are solved if all conflict clauses with at most 10 variables are kept and we only search up to the first fuiplevel. This corresponds to setting 'cf-111'. The number of best primal and dual bounds obtained are also in favor for this setting. Concerning the average number of nodes needed, the setting 'cf-1-11' performs much better on instance sets setS and setL but not on the Pack instances where 'cf111' is the best choice, also in terms of running

time on set setS, see Figures 2.18 and 2.19. Though 'cf010' is able to solve one more instance on set setL than the others, it needs much higher average running times on set setS and setL, more than twice as high on set setL.

Again, the tables show that there is no outperforming setting and much depends on the chosen branching decisions. But it can be observed that the more efforts are put into conflict analysis, the number of nodes can be often reduced to less than 80% and the average running time decreases by about one-half.

Impact of bound-widening

In the first setting we compute an explanation of minimum size for updates over maybe multiple points in time starting by the last peak to be reported and choosing largest possible steps back to the peak before. Similarly, we reimplemented the updating process by updating the lower bound by as few peaks as possible. Then, we only need to explain exactly one peak. The last setting uses the same approach but when computing an explanation it widens the bounds as much as possible. In a pure CP search this would not influence the solving process.

From Table 2.13 we can see that already the step-wise explanation needs on average less nodes though the time per node slightly increases. Using bound-widening techniques, the number of nodes decreases further, i.e., 50% on the Pack instances. For the other two test sets only a slight decrease in the number of nodes, about 10%, and running times can be found.

Table 2.13: Comparison of different strengths of explanations. 1: use minimum size set; 2: explain a a minimum size set peak-wise; 3: propagate peak-wise and explain a minimum-size set. A variable that is shifted over several peaks is reported more often. This seems to result in less nodes needed per instance in particular on instances where many nodes are needed.

setting	nopt	bprimal	bdual	gap	avtime	avnodes
setS (114 instances)					(allopt: 97 instances)	
cpsattt1	**99**	113	110	1.08	35.27	50048.99
cpsattt2	**99**	113	112	1.04	35.60	46015.84
cpsattt3	98	112	113	1.09	**28.97**	**42126.08**
setL (119 instance)					(allopt: 108 instances)	
cpsattt1	111	116	119	0.38	21.85	27051.62
cpsattt2	111	115	119	0.39	21.51	26586.49
cpsattt3	**113**	117	119	**0.32**	**16.87**	**21264.06**
Pack (55 instance)					(allopt: 35 instances)	
cpsattt1	39	52	54	1.70	9.62	32458.29
cpsattt2	38	51	55	1.74	6.59	16075.09
cpsattt3	39	52	54	1.72	**2.84**	**8524.86**

Conclusion and discussion

At this point we discuss the difference of the way conflict analysis works in SCIP in contrast to most SAT solvers like G12. SCIP performs its explanations backwards. That means if the problem becomes infeasible, the conflict graph is created from scratch starting with the infeasibility. In contrast, SAT solvers usually perform a depth-first search, their

branching tree is always a path, and they deliver the explanations on the fly whenever a bound change is performed. Hence, the information gathered is different in both solvers. A SAT solver updates the conflict scores immediately when a bound change is performed and hence also on variables that might not be involved in a conflict but are part of the implication graph that has stored all implications between the bound changes. In SCIP, the conflict graph is only set up as soon as an infeasibility is detected. Then, only bound changes that are related to that infeasibility will be given a higher conflict score. This might lead to worse scores in our solver.

Furthermore, in the model used by Schutt et al. [227] the conflict scores are calculated per variable and per value. Hence, the branching decisions are based on value scores per variable and are therefore more exact than ours. First experiments with adapted scores that are collected per value proved useful. A remark on this is given in Section 3.3.2.4.

2.4 A continuous relaxation of the cumulative constraint

Throughout the literature, it can be observed that CP approaches outperform IP approaches on most instance sets for RCPSP, especially when the makespan becomes large. The main reason is that the solving times of the LP relaxation take most of the running time. Until now, various IP formulations have been proposed that model the cumulative constraint. For an overview on exact formulations we refer to Koné et al. [164], who present time-indexed, event-based and flow-based integer programming formulations and experimentally evaluate the effectiveness of the gained lower bounds in a branch-and-bound search. They show that the event- and flow-based formulations are able to handle larger instances but the provided lower bounds are too weak to efficiently solve even small instances (up to 60 jobs). On the other hand, time-indexed formulations provide strong lower bounds but require high computation times.

In an exact IP formulation, an integer feasible solution to the LP relaxation yields a feasible schedule. On the other hand, inexact formulations can play an important role, as we will see here. Recall that the main roles of a relaxation are to provide dual bounds, to guide the search (e.g., branching on fractional variables) and to serve as an input for heuristics.

We propose a *continuous relaxation* of the cumulative constraint that is based on the integer start time variables and does not introduce any additional variables, in particular no binary variables. This relaxation supports the incompleteness of the propagation algorithms, i.e., it adds inequalities to the model when the propagation algorithms are no longer able to detect further bound adjustments. Hence, these inequalities are well suited to be implemented as a cutting plane procedure that is applied after the variable bounds are propagated. Let $J \subseteq \mathcal{J}$ be a set of jobs and S_j integer start time variables. The goal is to find valid inequalities of the form:

$$\sum_{j \in J} S_j \geq W, \tag{2.21}$$

The start times can be weighted by weights $w_j \in \mathbb{Q}$. Then, we are looking for inequalities of the form:

$$\sum_{j \in J} w_j S_j \geq W'. \tag{2.22}$$

Upper bounds on the sum of start times can be derived as well. Due to the symmetry, we will only consider the lower bound case here, whereas in the implementation we also generate these cuts to derive upper bounds on the sum of start time variables.

Related work Valid inequalities of that kind can be found by Queyranne and follow-up papers [122, 124, 211, 212] and Hooker and Yan [143]. Yalaoui and Chu [265] and Baptiste et al. [20] pursue a similar approach in order to derive dual bounds (not cutting planes) for parallel machine scheduling problems. Baptiste et al. evaluate the effectiveness of dual bounds derived from several relaxations in a branch-and-bound framework. But they do not consider the case where demands can be greater than one. We will relate our inequalities to these works in Section 2.4.2.3. Continuous relaxations have already been used by Jain and Grossmann [150] in a logic-based Benders decomposition framework, where the subproblem decomposes into a single machine scheduling feasibility problem with time windows. Thorsteinsson [248] shows that the continuous relaxation is crucial in their approach.

The search for valid inequalities of the form (2.21) and (2.22) can be described as a machine scheduling problem. Recall from Section 1.2 using the notation of the $\alpha|\beta|\gamma$-scheme [127][2], the capacity R corresponds to the number of machines m, the lower bounds est_j are release dates r_j, the upper bounds lct_j are deadlines d_j and the resource demands r_j are denoted by size_j. Then, checking whether a feasible solution to the cumulative constraint exists, corresponds to the problem $Pm|\mathsf{r}_j, \mathsf{d}_j, \mathsf{size}_j|-$. Now, using that the completion time C_j is given by $C_j = S_j + p_j$ (p_j is constant), finding a good lower bound W corresponds to $Pm|\mathsf{r}_j, \mathsf{d}_j, \mathsf{size}_j|\sum C_j$.

Next, we will review complexity results for that problem. After that, we present general separation procedures for the desired inequalities. Lower bounds on W in (2.21) are computed for $P|\mathsf{r}_j, \mathsf{size}_j|-$ and for a one-machine relaxation $1|\mathsf{r}_j, \mathsf{pmtn}|-$. Then, these procedures will be related to the aforementioned works. Finally, the effectiveness of the different procedures is evaluated in a computational study.

2.4.0.3 Complexity of $Pm|\mathsf{r}_j, \mathsf{d}_j, \mathsf{size}_j|\sum w_j C_j$

The hardness of $Pm|\mathsf{r}_j, \mathsf{d}_j, \mathsf{size}_j|\sum w_j C_j$ stems from the fact that most relaxations of the problem are already $\mathcal{NP}$-hard. Table 2.14 gives an overview on those problems related to $Pm|\mathsf{r}_j, \mathsf{d}_j, \mathsf{size}_j|\sum C_j$ that are polynomial solvable or $\mathcal{NP}$-hard.

Taking job sizes (demands) size_j into account is already strongly $\mathcal{NP}$-hard for $P|p_j = 1, \mathsf{size}_j|\sum C_j$ [97]. Scheduling on a single machine with release times $1|\mathsf{r}_j|\sum C_j$ is also strongly $\mathcal{NP}$-hard, see [171]. Such problems become easier if we allow preemption. Whether a feasible schedule for $Pm|\mathsf{r}_j, \mathsf{d}_j, \mathsf{pmtn}|-$ exists, can be decided in $O(n^3)$ using flow algorithms [142]. For $1|\mathsf{d}_j|\sum C_j$ the Smith-rule (scheduling last the job with largest available processing time) [240] runs in polynomial time and Baker [14] shows that $1|\mathsf{r}_j, \mathsf{pmtn}|\sum C_j$ can be solved in polynomial time by the shortest remaining processing time rule (SRPT-rule). In contrast, even if we allow preemption, the problem $1|\mathsf{r}_j, \mathsf{d}_j, \mathsf{pmtn}|\sum C_j$ is weakly $\mathcal{NP}$-hard [100] (but no pseudo-polynomial time algorithm has been given yet) and $P|\mathsf{r}_j, \mathsf{pmtn}|\sum C_j$ is strongly $\mathcal{NP}$-hard [18]. A weighted

[2] We abuse notation in this section to be consistent with the notation from machine scheduling, see Section 1.2. We use the term m machines instead of capacity R, deadline d_j for latest completion times lct_j and the resource demands r_j are denoted by size_j. Lower bounds or earliest start times est_j correspond to release times r_j.

objective function turns most problems into strongly $\mathcal{NP}$-hard ones, e.g. $1|\mathsf{r}_j, \mathsf{pmtn}|\sum C_j$ lies in P, but $1|\mathsf{r}_j, \mathsf{pmtn}|\sum w_j C_j$ is strongly $\mathcal{NP}$-hard [167]. To conclude, finding good bounds on W or W' is a challenging task on its own.

We point out that the hardness results for the weighted version holds for arbitrary weights, the case $w_j = p_j$ seems open. Similarly, hardness for the multi-machine case where all release dates are smaller than the earliest completion time is open – a case that is of practical interest in our algorithms.

	$\mathcal{NP}$-hard problems	polynomial solvable
single machine	$1\|\mathsf{r}_j, \mathsf{d}_j, \mathsf{pmtn}\,\|\sum C_j$ [100]	$1\|\mathsf{r}_j, \mathsf{d}_j, \mathsf{pmtn}\,\|-$ [142]
	$1\|\mathsf{r}_j\|\sum C_j$(s) [171]	$1\|\mathsf{r}_j, \mathsf{pmtn}\|\sum C_j$ [14]
		$1\|\mathsf{d}_j\|\sum C_j$ [240]
weighted	$1\|\mathsf{r}_j, \mathsf{pmtn}\|\sum w_j C_j$(s) [167]	$1\|\|\sum w_j C_j$
$m > 1$ machines	$P2\|\mathsf{r}_j, \mathsf{pmtn}\|\sum C_j$ [99]	$Pm\|\mathsf{r}_j, \mathsf{d}_j, \mathsf{pmtn}\|-$ [142]
	$P\|\mathsf{r}_j, \mathsf{pmtn}\|\sum C_j$(s) [18]	
	$P2\|\mathsf{size}_j\|\sum C_j$ [170]	
	$P\|\mathsf{pmtn}, \mathsf{size}_j\|\sum C_j$ [97]	
	$P\|p_j = 1, \mathsf{size}_j\|\sum C_i$(s) [97]	
	open: $P2\|\mathsf{d}_j, \mathsf{pmtn}\|\sum C_j$	

Table 2.14: $\mathcal{NP}$-hard and polynomial solvable problems closely related to $Pm|\mathsf{r}_j, \mathsf{d}_j, \mathsf{size}_j|\sum w_j C_j$ presented in $\alpha|\beta|\gamma$-scheme [127]. An (s) indicates strong $\mathcal{NP}$-hardness.

2.4.1 Separation procedure and an example

We will evaluate different separation procedures and different algorithms to derive lower bounds on $Pm|\mathsf{r}_j, \mathsf{d}_j, \mathsf{size}_j|\sum C_j$. First, we describe the following three separation procedures before we present the algorithms that actually compute the bounds on W.

1. All jobs $\mathcal{J}$ are relaxed to a single machine scheduling problem.
2. An earliest start schedule with respect to the lower bounds and without resource constraints is built. For each point in time $t \in \{\text{est}_j\}_j$ where the resource capacity is exceeded, the bound W is computed.
3. An earliest start schedule is build without resource constraints for all jobs that are scheduled at their earliest start time in the LP solution. Again, only a necessary subset of the earliest start times is considered to compute W.

Now, we go into more details. A first attempt is to relax the whole problem to a single-machine environment. Then, we use lower bounds derived from $1|\mathsf{r}_j, \mathsf{pmtn}\,|\sum C_j$ to identify a subset of jobs $J \subseteq \mathcal{J}$ that violate inequality (2.21).

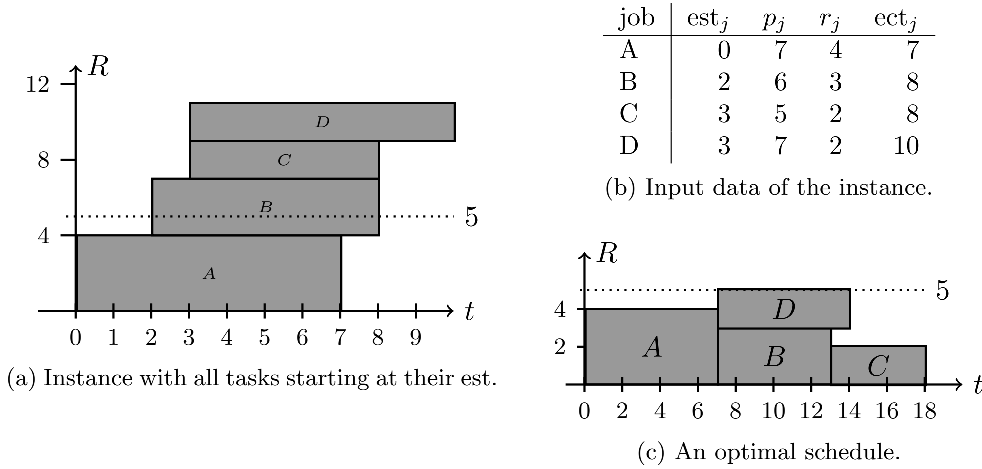
In the second approach, we evaluate an earliest start schedule, i.e., we assume all jobs start at their earliest start time. Let $J_t := \{j \in \mathcal{J} \mid \text{est}_j \leq t < \text{est}_j + p_j\}$ be the set of jobs running at time t in this schedule. Clearly, we only consider points in time $t \in \{\text{est}_j\}_j$, as our relaxations directly operate on the earliest start times. If $\sum_{j \in J_t} r_j > R$ holds, the capacity constraint is violated in this schedule and an amount W can be computed by which the jobs must be shifted away from their lower bounds. This bound can be

computed by a single machine relaxation or by packing arguments, the former will be denoted by W^{1m}, the latter by W^{size}.

In the third approach, we consider an earliest start schedule obtained by all jobs that are still at their lower bounds in the LP solution $\boldsymbol{S}^\star$ and use the same procedure as in the second case for that restricted set of jobs, i.e., $J_t := \{j \in \mathcal{J} \mid \mathrm{est}_j \leq t < \mathrm{est}_j + p_j \text{ and } S^\star_j = \mathrm{est}_j\}$. By that refinement, we often catch those jobs that are on a critical path and might not be shifted in the LP solution by the second procedure.

The following example will be used to illustrate the computation of the different lower bounds on W. Recall that upper bounds on the start times are computed in our experiments by symmetric arguments.

Example 2.5. We are given four jobs as depicted in Figure 2.22 using one resource with capacity 5. Figure 2.22a shows an earliest start schedule, the input data in Figure 2.22b and an optimal schedule, see Figure 2.22c, for $Pm|\mathsf{r}_j, \mathsf{size}_j|\sum C_j$. A trivial lower bound on the sum of start times is given by $\sum_j \mathsf{r}_j = \sum_j \mathrm{est}_j = 8$, whereas a maximum lower bound on the sum of start times is given by 27, which is obtained from the optimal solution.



(a) Instance with all tasks starting at their est.

job	est_j	p_j	r_j	ect_j
A	0	7	4	7
B	2	6	3	8
C	3	5	2	8
D	3	7	2	10

(b) Input data of the instance.

(c) An optimal schedule.

Figure 2.22: Illustration of example 2.5 with 4 jobs.

2.4.2 Relaxations and bounding techniques

2.4.2.1 Lower bounds from the single-machine relaxation W^{1m}

An instance of $Pm|\mathsf{r}_j, \mathsf{size}_j|\sum C_j$ can be relaxed to $1|\mathsf{r}_j, \mathsf{pmtn}|\sum C_j$ by keeping the release dates r_j as they are and setting the processing times to $p_j \cdot \mathsf{size}_j/m$. This way, we compute a lower bound W^{1m} by scheduling the jobs preemptively according to SRPT-rule for $1|\mathsf{r}_j, \mathsf{pmtn}|\sum C_j$. In order to improve the bound, we complement the obtained completion times with the i-th smallest completion time, see [20] for a similar approach in parallel machine scheduling where a proof of correctness has already been given.

Lemma 2.29. *Let* $\mathrm{ect}[j]$ *be the* j*-th smallest earliest completion time among all jobs and* $C^{\mathsf{pmtn}}[j]$ *be the* j*-th smallest completion time obtained in the preemptive schedule*

of $1|\mathsf{r}_j, \mathsf{pmtn}|\sum C_j$. *Then,*

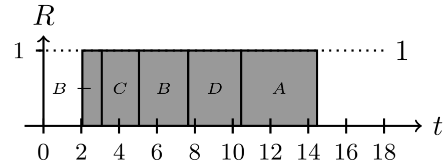
$$W^{1m} := \sum_j \max\{C^{\mathsf{pmtn}}[j], \mathrm{ect}[j]\} - \sum_j p_j$$

is a valid lower bound on W *in* (2.21).

Later, we use this relaxation in all three separation procedures. In the implementation, we consider some special cases for the three separation routines (relaxing all jobs, only est-peaks and only LP-est-peaks). In case that we relax all jobs at once, then in the preemptive schedule obtained from SRPT, holes may occur, if the release time of a job is larger than the latest completion time of all jobs with smaller release times in the preemptive schedule. In this case, we treat these jobs separately and only generate cuts on these subsets. In case that we only look at peaks of an earliest start schedule, we additionally adjust the release times of the single machine relaxation, i.e., we compute t_0, the point in time at which the jobs first exceed the capacity. The release times r_j are adjusted to $\mathsf{r}'_j := \max\{\mathsf{r}_j, t_0\}$. Then, for each job we create a machine-job with release time r'_j and processing time $p'_j := \min\{p_j, p_j - (t_0 - \mathsf{r}_j)\}\mathsf{size}_j/m$ and run SRPT plus ect-adjustment. This is exemplarily shown in the following example.

	r_j	p_j	$\lceil C_j \rceil$	C^{ect}
A	2	$5 \cdot 4/5 = 4$	15	15
B	2	$6 \cdot 3/5 = 3.6$	8	8
C	3	$5 \cdot 2/5 = 2$	5	7
D	3	$7 \cdot 2/5 = 2.8$	11	11
			$\sum = 39$	$\sum = 41$

(a) Calculations for W^{1m}. C^{ect} are the completion times after adjustment.



(b) Solution of the preemptive relaxation.

Figure 2.23: Computation of W^{1m} for example 2.5.

Example 2.6 (Example 2.5 continued)**.** When splitting the set of jobs at $t = 3$, we obtain a one machine relaxation with release times, processing times and optimal completion times as tabulated in Figure 2.23a according to a preemptive schedule obtained by the Smith rule, see Figure 2.23b.

Without adjustment, we obtain $W \geq 39 - 25 = 14$, whereas after the adjustment, we get $W^{1m} = 41 - 25 = 16$. Altogether, this yields the inequality $S_A + S_B + S_C + S_D \geq 16$.

We remark that job A is splitted at $t = 2$ where the first demand peak exceeds the capacity such that A only needs to be processed for 5 units of time after $t = 2$. Nevertheless, its processing time of 7 needs to be respected when W^{1m} is computed.

Further adjustments prior to computing the SRPT schedule turned out to be effectively. In particular, strengthening the coefficients, see Section 3.2.1, for such small sets of jobs turns out to be no computational bottleneck at all and often yields much better bounds. In our example, since job A with $\mathsf{size}_A = 4$ cannot be scheduled in parallel to any other job, we can strengthen its resource demand to $\mathsf{size}_A := 5$ which yields a processing time of $5 \cdot 5/5 = 5$ instead of 4. Without earliest completion time adjustment, we obtain the lower bound $W \geq 15$ which is already better than the initial value of 14.

Example 2.7 (Example 2.6 continued)**.** For $t = 2$ an optimal bound is obtained on the jobs A and B when coefficient strengthening is used: As A and B cannot be scheduled in parallel, we can relax the problem to a single machine relaxation and obtain $S_A + S_B \geq W^{1m} = 7 + 13 - (7 + 6) = 7$. Whereas without coefficient strengthening in the single machine environment we calculate with $p_A = 4$ and $p_B = 18/5$, and only obtain after ect-adjustment: $W^{1m} = 17 - 13 = 4$.

To conclude, for the refinements of the single machine relaxation we learned the following. Adjustments of the earliest completion times are helpful if jobs have high processing times but small demand as this induces a low energy requirement in the single machine relaxation. Coefficient strengthening works particularly well if jobs are disjunctive, because holes in the capacity profile are filled this way.

2.4.2.2 Lower bound W^{size} based on $Pm|\mathsf{r}_j, \mathsf{size}_j|-$

In this relaxation, we consider sets of jobs $\mathcal{J}$ that are pairwisely scheduled in parallel in an earliest start schedule while the sum of their demands exceeds the capacity. We call them *concurrent jobs*. This set can be characterized as follows: for each pair (i, j) of jobs $i, j \in \mathcal{J}$, $\mathsf{r}_j + p_j \geq \mathsf{r}_i$ and $\mathsf{r}_i + p_i \geq \mathsf{r}_j$ hold. The following inequalities take the different sizes of the concurrent jobs into account and delay those that cannot be scheduled first behind the lowest earliest completion time.

Lemma 2.30. *Let $\mathcal{J}$ be a set of n concurrent jobs. For $J \subseteq \mathcal{J}$ we set* $\text{ect}(J) := \min_{j \in J}\{\mathsf{r}_j + p_j\}$. *Then, we have the following lower bound on W:*

$$W^{\mathsf{size}} := \min_{J \subseteq \mathcal{J} : \sum_{j \in J} \mathsf{size}_j \leq m} \left\{ \sum_{j \in J} \mathsf{r}_j + (n - |J|) \cdot \text{ect}(J) \right\}.$$

Proof. On the right hand side of the inequality, we compute the minimum over all sets J. Now we prove the lemma statement by showing that this inequality holds for any feasible solution. Let S' be a feasible solution to $P|\mathsf{r}_j, \mathsf{size}_j|$. Let $J \subseteq \mathcal{J}$ be a set of jobs that are scheduled first, i.e., for all $j \in \mathcal{J} \setminus J : S^*_j \geq \min_{k \in J}\{S^*_k + p_k\}$ and for $j, k \in J, j \neq k$: $S'_j + p_j > S'_k$. Then: $\sum_{j \in J} S'_j \geq \sum_{j \in J} \mathsf{r}_j$ and $\sum_{j \in \mathcal{J} \setminus J} S'_j \geq \min_{k \in J}\{S'_k + p_k\} = \text{ect}(J)$. This yields:

$$\sum_{j \in \mathcal{J}} S'_j = \sum_{j \in J} S'_j + \sum_{j \in \mathcal{J} \setminus J} S'_j \geq \sum_{j \in J} \mathsf{r}_j + (n - |J|) \cdot \text{ect}(J).$$

□

This bound is weak at first sight, but helpful in our separation procedure. Certainly, only inclusion maximal sets J need to be considered in the minimization. Instead of checking all inclusion maximal sets J, we use a lower bound in which we sort the job sizes size_j in non-decreasing order and find a valid minimum delay for the sum over all jobs.

Theorem 2.31. *Given a set of concurrent jobs $\mathcal{J}$. Let $\mathsf{size}[j]$ be a sorting of non-decreasing sizes size_j, and $\mathsf{r}[j]$ a sorting of the release times r_j in non-decreasing order. Let $k' := \text{argmax}_k\{\sum_{j=1}^{k} \mathsf{size}[j] \leq m\}$. Then:*

$$W^{\mathsf{size}_1} = \sum_{j=1}^{k'} \mathsf{r}[j] + (n - k') \min_j\{\mathsf{r}_j + p_j\}.$$

Proof. Recall the notation and the proof of Lemma 2.30. Let J be a set of jobs for which the minimum is attained. The case $k' > |J|$ is impossible, as k' is the maximum number of jobs that can be scheduled in parallel, hence in J at most k' jobs can be scheduled at their release time. For $k' \leq |J|$ we use the fact that $\mathrm{ect}(J) \geq \mathrm{ect}(\mathcal{J}) = \min\{\mathsf{r}_j + p_j\}$ and $\mathsf{r}[j] < \mathrm{ect}(J)$ as all jobs are concurrent and calculate:

$$\sum_{j \in J} \mathsf{r}_j + (n - |J|) \cdot \mathrm{ect}(J) \geq \sum_{j=1}^{k'} \mathsf{r}[j] + (n - k') \min_j\{\mathsf{r}_j + p_j\}.$$

□

The correctness of Theorem 2.31 is pretty intuitive: The first summand contains the k' smallest release dates, where k' is the largest number of jobs that can be scheduled at their release times, if these jobs would have the k' smallest resource demands. All others must be postponed after the smallest earliest completion time $\min_j\{\mathsf{r}_j + p_j\}$.

Example 2.8. For point in time $t = 3$, we obtain $\mathsf{r}[1] = 0, \mathsf{r}[2] = 2, \mathsf{r}[3] = \mathsf{r}[4] = 3$, and $\mathsf{size}[1] = \mathsf{size}[2] = 2, \mathsf{size}[3] = 3, \mathsf{size}[4] = 4$; then $k' = 2$. Since at most two tasks can be executed in parallel (the two smallest release times are $0, 2$) and two tasks must be postponed after the earliest completion time. We calculate:

$$W^{\mathsf{size}_1} = \sum_{j=1}^{2} \mathsf{r}[j] + (n - k') \cdot \min_j\{\mathsf{r}_j + p_j\} = (0 + 2) + (4 - 2) \cdot \min\{7, 8, 8, 10\} = 16.$$

Hence, $S_A + S_B + S_C + S_D \geq 16$. Equivalently, at $t = 2$ we obtain the valid inequality $S_A + S_B \geq 7$.

If the set of concurrent jobs is large and only few can be scheduled in parallel we give an improved lower bound. Knowing that all jobs in J are available at time t as in our separation procedure, let $N := \max_{J \subseteq J_t}\{|J| \mid \sum_{j \in J} \mathsf{size}_j \leq m\}$. Then, these jobs can be scheduled in at least $B := \lceil \frac{n}{N} \rceil$ groups of at most N jobs each. We conclude that the first N jobs can be scheduled at the N smallest release times. The next N jobs in the i-th $(i = 2, \ldots, B - 1)$ block can be scheduled at the $(i - 2)N + 1$-th earliest completion time, since $(i - 2)N$ jobs have been scheduled two blocks before. A job of the i-th block can start after the first job of the $i - 1$-st block finishes. In the last block $i = B$ only $n - \lceil \frac{n}{N} \rceil N$ jobs are scheduled after the $(B - 2)N + 1$-st earliest completion time. This observation is condensed in the following corollary.

Corollary 2.32. *Given a set $J \subseteq \mathcal{J}$ of concurrent jobs. Let $\mathsf{r}[j]$ be a sorting of the release times in non-decreasing order and denote by $\mathrm{ect}[i]$ the i-th smallest earliest completion time among $\{\mathsf{r}_j + p_j\}_{j \in \mathcal{J}}$. Then, a valid lower bound on W in (2.21) is given by:*

$$W^{\mathsf{size}_2} \geq \sum_{j=1}^{N} \mathsf{r}[j] + N \cdot \left(\sum_{i=2}^{B-1} \mathrm{ect}[(i-2)N + 1] \right) + (n - N\lfloor \frac{n}{N} \rfloor) \cdot \mathrm{ect}[(B-1)N + 1].$$

2.4.2.3 Relation to other known valid inequalities

We presented different bounds on W in (2.21). Next, we discuss the relation to similar results from the literature. We start with a presentation of inequalities by Queyranne [212].

Next, Hooker and Yan's [143] inequalities are presented. Afterwards, we relate our lower bounds from the single machine relaxation to those that have been used in the context of parallel machine scheduling, see e.g. [20, 265].

Queyranne's valid inequalities One of the first valid inequalities presented in the literature that focuses only on the start time variables (without binary variables) are the *shifted parallel inequalities* [122, 124, 211, 212]. They hold for the one machine case with release times and arbitrary processing time. We denote by $\mathsf{r}_{\min}(J) = \min_{j \in J}\{\mathsf{r}_j\}$ and by $p(J) = \sum_{j \in J} p_j$.

Lemma 2.33 ([122, 124, 211, 212])**.** *In a* $1|\mathsf{r}_j|-$ *environment, for any set of jobs* $J \subseteq \mathcal{J}$ *and any preemptive schedule with completion times* C_j*, the following inequality holds*

$$\sum_{j \in J} p_j C_j \geq p(J)\mathsf{r}_{\min}(J) + \frac{1}{2}\left(\sum_{j \in J} p_j^2 + (\sum_{j \in J} p_j)^2\right), \tag{2.23}$$

and equality holds if and only if all jobs in J *are scheduled without interruption from* $\mathsf{r}_{\min}(J)$ *to* $\mathsf{r}_{\min}(J) + p(J)$*.*

We remark here that Goemans et al. [124] stated the lemma for the mean busy times M_j for which $M_j \leq C_j - p_j/2$ holds. Then, the authors show that a list scheduling algorithm, using the weighted shortest remaining processing time rule (WSRPT, preemptive Smith rule [240]), yields an optimal solution for

$$\min\left\{\sum_{j \in \mathcal{J}} w_j(M_j + \frac{1}{2}p_j) \mid \sum_{j \in J} p_j C_j \geq p(J)\mathsf{r}_{\min}(J) + \frac{1}{2}\left(\sum_{j \in J} p_j^2 + (\sum_{j \in J} p_j)^2\right), \forall\, J \subseteq \mathcal{J}\right\}.$$

Observe that this list schedule derives an optimal solution for $1|\mathsf{r}_j, \mathsf{pmtn}\,|\sum w_j M_j$ and due to $C_j \geq M_j + p_j/2$ it provides a lower bound for the problem $1|\mathsf{r}_j, \mathsf{pmtn}\,|\sum w_j C_j$ and hence also for $1|\mathsf{r}_j|\sum w_j C_j$. The separation problem to inequalities (2.23) can be solved in $O(n^2)$ [211] by trying all possible values for $\mathsf{r}_{\min}(J)$ (at most n) and finding a maximum violating set, see [210]. Nevertheless, these inequalities are not facet-defining in general. In our context, the bounds generated are quite weak after scaling from a RCPSP environment when applied to all jobs $\mathcal{J}$. That's why, we apply a separation procedure to identify a promising subset of jobs.

Cuts presented by Hooker and Yan A special class of facet-defining inequalities has been introduced by Hooker and Yan [143]. They give valid inequalities for the case where all start time variables have the same lower bound. Their lower bound is related to a bin-packing relaxation. These inequalities are facet-defining for $Pm|\mathsf{size}_j = \mathsf{size}, p_j = p|-$. The authors also present (not facet-defining) inequalities for the more general case $Pm|\mathsf{size}_j|-$.

Given a set of jobs $\{j_1, \dots, j_k\}$. Each of the jobs is split into $n_{j_i} = \lfloor p_{j_i}/\Delta \rfloor$ segments of equal duration $0 < \Delta \leq \min_i\{p_{j_i}\}$. Excess $\mathsf{size}_{j_i} > \Delta\lfloor \mathsf{size}_{j_i}/\Delta \rfloor$ is ignored. Let $k' = \sum_{i=1}^{k} n_{ji}$ be the total number of segments, each segment j of job j_i has demand of $\mathsf{size}'_j = \mathsf{size}_{j_i}$ and weight $w_j = 1/n_{j_i}$. Then the following theorem holds.

Theorem 2.34 (Hooker and Yan [143])**.** *Given any set of jobs* $\{j_1, \ldots, j_k\}$*, the inequality* $S_{j_1} + \ldots + S_{j_k} \geq h_{relax}$ *is valid for* $Pm|\mathsf{size}_j|-$*, where*

$$h_{relax} = \sum_{j=1}^{k'} w_j \left(q_j - 1 - (q_j - q_{j-1}) \frac{(q_j - 1)m - Q_{j-1}}{\mathsf{size}'_j} \right) - E$$

and

$$q_j = \lfloor \frac{Q_j}{m} \rfloor + 1 \qquad Q_j = \sum_{\ell=1}^{j} \mathsf{size}'_\ell \qquad E = \sum_{i=1}^{k} \frac{1}{2}(n_{j_i} - 1)\Delta,$$

where $\Delta \in [0, \min_i\{p_{j_i}\}]$.

A line search is performed to determine the best value for Δ. The maximum on h_{relax} occurs at a value of Δ that evenly divides at least one of the processing times p_{j_i}, or zero. In case $\Delta = 0$, the inequality simplifies to:

Corollary 2.35. *Renumber the jobs* $j_1, \ldots, j_k$ *using indices* $q = 1, \ldots, k$ *so that the products* $e_q = \mathsf{size}_q \cdot p_q$ *occur in non-decreasing order. The following is a valid inequality for the problem, when* $\Delta \to 0$*:*

$$S_{j_1} + \ldots + S_{j_k} \geq \sum_{q=1}^{k} ((k - q + \frac{1}{2}) \frac{\mathsf{size}_q}{m} - \frac{1}{2}) p_q.$$

In particular, the last corollary is a simplification of Smith's SPT-rule, that can be applied here since all release dates are relaxed to zero (there exists an optimal non-preemptive schedule). The same holds for the value of h_{relax} in the theorem. The first summand in brackets computes for each j it's completion time, assuming that $j - 1$ segments have already been scheduled. Since each job was partitioned into n_{j_i} segments, the weighting of this summand with w_j yields the average of these completion times, hence a value of at most $C_j - p_j/2$. In order to obtain a valid bound on the start times, the correction term E, subtracts half of the processing times from each job. To the best of our knowledge, it has not yet been possible to extend these inequalities successfully to non-equal release times.

Relation to work on m-machines Baptiste et al. [20] present lower bounds for $Pm|\mathsf{r}_j|-$ with different objective functions. They derive heuristic, combinatorial and exact approaches. Some of the bounds have also been derived by Yalaoui and Chu [265] for $Pm|\mathsf{r}_j|\sum C_j$. The exact relaxations (based on IP, Lagrangean relaxation,...) as both paper show in their experiments are too time-consuming and hence not applicable to large instances as a subroutine throughout branch-and-bound search. Hence, they use combinatorial lower bounds in order to prune unpromising nodes. No cutting planes are generated in these works. Their combinatorial relaxations are based on:

(i) Relaxing all release times to the smallest one or a value in between,

(ii) On *job-splitting*: preemptively schedule the job according to SRPT-rule and more than one unit of a job can be processed at a point in time,

(iii) Horn's theorem [145] for solving $Pm|\mathsf{r}_j, \mathsf{pmtn}|C_{\max}$, and

(iv) Combinations of these with improvement to the i-th smallest completion time.

These approaches differ from ours in the assumption that all jobs have uniform resource demand. Our single machine relaxation implicitly contains the splitting of the jobs (since we only have one machine and allow preemption and maybe more than size_j units can be processed at one point in time). Their usage of Horn's theorem as a relaxation for the $\mathcal{NP}$-hard problem $Pm|\mathsf{r}_j, \mathsf{pmtn}|\sum C_j$ is obsolete in the single machine case since $1|\mathsf{r}_j, \mathsf{pmtn}|\sum C_j$ can be solved in polynomial time, as in our lower bound W^{1m} presented. An open question remains whether the capacity demands of the jobs can be closer incorporated into the works [20, 265] in order to derive better or similar bounds as presented here.

2.4.3 Computational study

X	separation method	y	relaxation
1	1-machine	a	W^{1m}
2	est and sol peaks	b	W^{size_1}
3	est peaks	c	W^{size_2}
4	sol peaks	d	$W^{1m}, W^{\mathsf{size}_2}$
6	standard IP		

Table 2.15: Encoding of different settings used for the continuous relaxation denoted by "sepaDXy".

In this section, we evaluate the presented separation procedures for our continuous relaxation in a branch-and-bound framework on RCPSP instances. Recall the standard tables and figures from Section 1.5.3, where time is given in seconds and gaps in percentages.

We start by describing the settings. We compare our continuous relaxation to the results obtained with a MIP model as presented in Section 1.3.1, denoted by "sepaD6", and a pure CP-SAT hybrid "sepaCPSAT". The different ways of separating the cuts, such as relaxing each cumulative constraint into a single machine problem, respecting all points in time where an earliest start schedule of all jobs violates the capacity, or only of those jobs where the LP start time corresponds to a lower bound. The different kinds of relaxations, i.e., single machine relaxation W^{1m}, or the packing relaxations $W^{\mathsf{size}_1}, W^{\mathsf{size}_2}$ and their combinations are also studied next. For ease of reading, we denote each setting by "sepaDXy", where 'X' corresponds to the separation method and 'y' to the relaxation. Both parameters are specified in Table 2.15.

Table 2.16 presents the numerical results. In general, none of the chosen settings outperforms the other. Hence, there is no final clue which method is best to use. We observe that all methods are slightly better than the pure CP-SAT approach and much better than using the standard IP formulation.

Figure 2.24 shows that the IP approach "sepaD6" is able to solve about 10% fewer instances than the other approaches and needs already after 50% of all instances more than 100 sec. to solve these, whereas all other approaches need more than 100 sec. only for 25% of the instances. Looking carefully into the graphics, the difference between the continuous relaxation and a CP-SAT approach is only small. But all indicators are favorable for the continuous relaxation as there are slightly more instances solved to

optimality if a continuous relaxation is used and the total running times are also slightly better.

In particular, there are several settings, where a continuous relaxations needs about half the number of search nodes and half the running time on average compared to a CP-SAT approach. On the other hand, on the Pack instances there are settings like 'sepaD2c' and 'sepaD3d' (in both the bound W^{size_2} is used), where the number of nodes drastically increases. This might be due to the fact that these instances are highly cumulative and energetic propagators should be used. Next, we evaluate the impact of the branching rule to the results observed here. There, we use an energy based propagation rule, i.e., time-table edge-finding, on the Pack instances and we will see that by doing so, the continuous relaxation becomes better compared to the CP-SAT approach.

Having this huge amount of different kinds of separation techniques and different relaxations, looking at one-to-one comparison of running time and number of nodes (not presented here), there is no advice we can give which combination is the best. It seems that a pure single machine relaxation, W^{1m}, provides too weak bounds and not many nodes can be fathomed this way. On the other hand, sometimes the strongest relaxation due to the job sizes (W^{size_2}) needs too much computation time. In the next experiments we separate cuts on earliest start time peaks and peaks of the LP solution using the bounds W^{1m} and W^{size_2} which corresponds to setting 'sepaD2d' and makes a fair tradeoff between running time, the number of best primal and dual bounds obtained and the number of optimally solved instances.

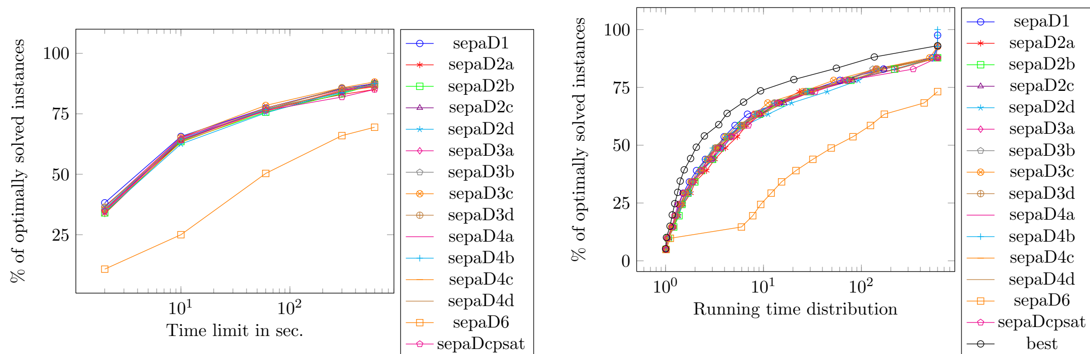


Figure 2.24: Number of optimally solved instances for different kinds of continuous relaxations.

Impact of pseudocost branching When solving a linear relaxation of the problem throughout search, an LP solution is available in each node of the search tree. This can be used to guide the branching decisions, e.g., by performing reliable pseudocost branching, i.e., pseudocosts per variable are initialized by strong branching if a variable is a fractional branching candidate and has not been branched on so far or for a longer time. See Section 3.3 for more details on this. As our relaxation is rather weak, not the pruning capabilities of the relaxation should be most helpful but rather different scores on the variables that are used as branching decisions or the way the domains of the integer variables are split. To see whether this is true or not, we next compare the CP-SAT approach against the continuous relaxation with and without reliable pseudocost branching in order to evaluate the impact of this branching scheme.

Table 2.16: Comparison of different continuous relaxations with a CP-SAT and a standard IP formulation.

setting	nopt	bprimal	bdual	gap	avtime	avnodes
setS (114 instances)					(allopt: 79 instances)	
sepaD1	101	101	109	0.80	5.55	8575.43
sepaD2a	99	99	105	1.06	6.41	7531.14
sepaD2b	**102**	103	110	0.82	4.85	5986.67
sepaD2c	101	102	107	0.89	5.14	6244.08
sepaD2d	100	101	109	0.92	5.34	6506.35
sepaD3a	98	98	108	1.04	5.64	7707.10
sepaD3b	101	101	109	0.86	4.87	6901.25
sepaD3c	101	101	110	0.82	4.87	6917.54
sepaD3d	98	98	107	1.02	5.23	7005.34
sepaD4a	100	100	107	0.92	4.82	6862.11
sepaD4b	101	102	108	0.78	4.39	5941.84
sepaD4c	**102**	103	110	0.77	4.27	6038.71
sepaD4d	101	102	110	0.82	**4.08**	**5772.57**
sepaD6	79	94	79	5.38	76.98	9370.47
sepaDcpsat	97	97	102	1.21	6.23	10001.22
setL (119 instance)					(allopt: 76 instances)	
sepaD1	114	114	118	0.30	5.50	5410.59
sepaD2a	111	111	116	0.56	3.43	2489.16
sepaD2b	113	114	116	0.41	3.19	**2216.16**
sepaD2c	**115**	116	117	0.24	2.94	2234.49
sepaD2d	114	114	117	0.32	3.18	2228.96
sepaD3a	111	111	116	0.50	4.83	3674.32
sepaD3b	112	112	118	0.48	3.29	2601.78
sepaD3c	114	114	118	0.39	**3.00**	2501.76
sepaD3d	112	112	118	0.47	3.30	2620.49
sepaD4a	113	113	117	0.41	3.04	2610.36
sepaD4b	115	115	118	0.26	4.54	3589.41
sepaD4c	113	113	117	0.44	5.25	3907.95
sepaD4d	114	114	118	0.31	4.63	3831.45
sepaD6	81	91	83	3.80	64.17	3144.25
sepaDcpsat	109	109	115	0.53	7.16	6404.54
Pack (55 instance)					(allopt: 30 instances)	
sepaD1	38	41	54	1.72	1.95	3743.93
sepaD2a	36	39	54	1.81	1.51	2021.23
sepaD2b	37	40	53	1.74	**1.39**	**1458.77**
sepaD2c	38	42	53	1.66	6.31	14790.47
sepaD2d	38	42	52	1.59	1.98	3515.00
sepaD3a	**40**	43	53	1.61	1.76	3329.17
sepaD3b	38	41	53	1.73	3.26	7269.33
sepaD3c	**40**	43	53	1.60	**1.35**	1631.23
sepaD3d	**40**	43	53	1.64	7.16	14041.40
sepaD4a	37	40	54	1.68	1.50	2205.07
sepaD4b	37	40	54	1.88	1.53	2491.33
sepaD4c	34	37	54	1.92	4.38	9990.07
sepaD4d	37	40	**55**	1.67	2.06	3812.87
sepaD6	**40**	**51**	52	**1.33**	8.05	3127.80
sepaDcpsat	**40**	43	53	1.65	3.04	7184.13

Figure 2.26 shows the percentage of optimally solved instances per instance set and the percentage of instances solved within different time limits. We observe that a CP-SAT approach solves fewer instances to optimality of the whole set than a continuous relaxation. For the Pack instances, we point out that it is crucial to use the time-tabling edge-finding propagator and not only time-tabling. Without this propagator on the Pack instances, the CP-SAT approach performs by far best. This can be easily explained.

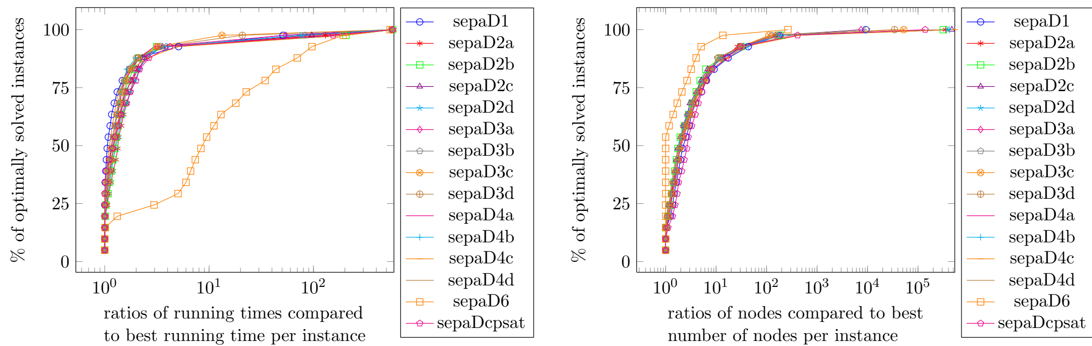
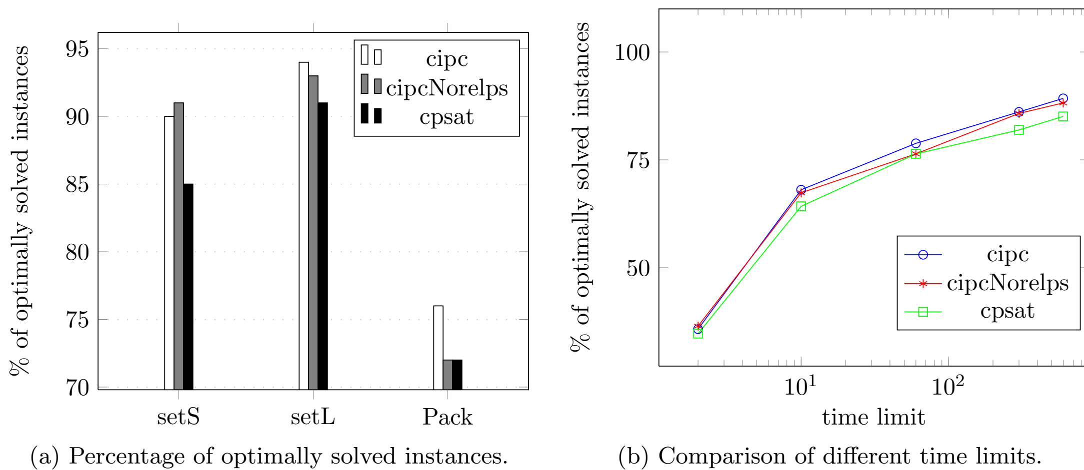


Figure 2.25: Ratios of nodes and running times given for all optimally solved instances from sets setS, setL and Pack obtained by using different continuous relaxations compared to a CP-SAT approach and a standard IP formulation.



(a) Percentage of optimally solved instances.

(b) Comparison of different time limits.

Figure 2.26: Percentage of optimally solved instances using a CP-SAT hybrid, or the continuous relaxation with and without reliable pseudocost branching.

When the propagation algorithms are no longer able to update variable bounds, but some jobs are running concurrently, the continuous relaxation provides a mean to tell the framework that these concurrent jobs must be shifted. If no energetic arguments are used to tighten the variable bounds, the information provided by the continuous relaxation is rather poor and misleading.

To conclude, the reliable pseudocost branching rule has only a small impact on the solving process. There are more instances solved on instance sets setL and Pack with this rule and on instance set setS fewer instances as well. We rather assume that the slightly better performance stems from the different domain splitting schemes that are applied, if an LP solution is available. Without an LP relaxation the SCIP internal branching rules split the domain of a variable in two equally sized domains. Whereas given an LP solution, the domain is split in two sections that are determined by the LP solution value. The branching variable itself is chosen according to a hybrid score, again we refer to Section 3.3.1 for further discussion.

Discussion and conclusion In order to detect promising search spaces within reasonable running time, and thereby make good branching decisions, our results are slightly in favor of using a continuous relaxation. That way, pseudocost values can be used as branching decisions and the domain of the variables can be split according to the LP solution values.

Latest experiments with SCIP show that collecting branching scores not only per variable but also per value leads to smaller branching trees, mainly due to better branching decisions. This idea is worth-while to be transferred to MIP solving where pseudocost per variable are collected. This may even improve on our results presented here. A remark on our preliminary experiments can be found in Section 3.3.2.4.

Chapter 3

Lower bounds, presolving techniques and branching schemes for RCPSP

Propagation techniques that reduce the domains of the variables are important to solve hard combinatorial problems efficiently, as they shrink the size of the problem and usually lead to smaller branching trees. Of similar importance in a branch-and-bound framework is to efficiently compute strong dual bounds to prune unpromising search spaces early. Several of such techniques are nowadays well established for RCPSP, e.g., several lower bounds are known, adding redundant constraints (disjunctive and cumulative ones) using transitive precedence relations, different IP formulations and valid inequalities are key techniques in this area.

In this section we first introduce well known lower bounds for RCPSP in Section 3.1. These bounds give a lower bound on the makespan for a particular set of jobs with their domains. Hence, this lower bound can also be used when other objective functions than makespan minimization are considered. These bounds can also be applied to sub-networks and will be used during a presolving step to add new constraints or to strengthen transitive precedence relations, as described in Section 3.2.

In *presolving*, which is done prior to branch-and-bound search, several more complex propagation rules are executed. During this step, the constraints try to tighten their coefficients in order to strengthen the LP relaxation or to yield better bound adjustments throughout search. The problem may also be decomposed, or new constraints may be added in order to fasten the search later. We give an overview on presolving techniques, like adding redundant resources, and present coefficient strengthening techniques that adjust the demands and capacity of the cumulative constraint in order to improve the bound changes derived by edge-finding, energetic reasoning and time-table edge-finding.

We show that using an elaborate presolving pays off on all the given instances, as 21 more instances can be solved already within presolving due to better dual bounds. Some open instances from the Pack set can be closed for the first time within less than 10 min. By just applying coefficient strengthening, the dual gap in the root node, i.e., the difference of dual value in the root node minus the final dual value divided by the final dual value, decreases from 3.3% to 0.4% on average. Sub-networks can be detected on 1,129 of 2,095 instances. Applying all the proposed presolving techniques on the Pack instances and using the standard MIP formulation from Section 1.3.1 throughout search,

the running time decreases by one-half and the number of nodes decreases even to 10%.

In Section 3.3 we discuss scheduling specific and generic branching schemes for RCPSP. Given the presolving techniques derived in this section and the strengths from CP, IP and SAT, we evaluate these branching schemes and elaborate on the different strengths of the developed solving techniques. On PSPLib instances, a CP-SAT hybrid (often together with the continuous relaxation) and on Pack instances, a hybrid of CP, IP and SAT using the binary MIP formulation perform best. In particular for the Pack instances of which 69% are closed, we are able to increase this number to 78%.

3.1 Lower bounds

Lower bounds are used throughout search to cutoff infeasible nodes early. These lower bounds can be combinatorial, based on IP formulations or may stem from integrated approaches. Often they are also used as subroutines during presolving. We start with an overview of several lower bounds on the makespan $C_{\max}$ for RCPSP in Section 3.1.1 and present an efficient procedure to compute a preemptive bound. In our computational experiments this bound does not yield strong propagations throughout search. But applied to sub-networks as done in Section 3.2 it helps to tighten coefficients of transitive precedence relations.

3.1.1 Known lower bounds

Critical path bound A trivial lower bound on the makespan is given by a longest path calculation in the precedence network. The longest path can be calculated in $O(|E|)$ with a breadth-first search and is often called *critical path* (shifting any job on that path increases the makespan). This lower bound is denoted by $\text{LB}_0 := \max_j\{\text{ect}_j\}$.

Observe that the critical path is in general not unique. Stinson, Davis and Khumawala [245] improved this lower bound by considering all jobs that are not part of a critical path. For each critical path and each such job, they compute the maximum amount of time t_j, that this job can be executed in parallel along a critical path P from the set of all critical paths $\mathcal{P}$. They derive $\text{LB}_S := \text{LB}_0 + \max_{P\in\mathcal{P}} \max_{j\notin P}\{\max\{0, p_j - t_j\}\}$. This idea has been improved by Demeulemeester, see Brucker [38]), by augmenting a critical path via a second node-disjoint path. Then, a two-path relaxation is solved by dynamic programming.

Volume bound Another rather simple lower bound can be deduced by computing the total demand of all jobs of one resource. This demand is divided by the capacity and can be rounded up to the next larger integer value.

This lower bound is denoted by $\text{LB}_v(J) := \max_{r\in\mathcal{R}}\{\lceil\sum_{j\in J} r_j \cdot p_j/R_k\rceil\}$.

Disjunctive graph Here, we consider all pairs of jobs that cannot be executed in parallel due to some resource constraints. For each such pair (i, j), two subinstances are created in which temporarily either a precedence constraint $\texttt{precedence}(S_i, S_j, p_i)$ or $\texttt{precedence}(S_j, S_i, p_j)$ is added. For each subinstance the longest path in the precedence network is calculated and the minimum over both yields a feasible lower bound.

This bound has been used e.g. by Klein and Scholl [159]. Brucker et al. [39] improved this bound by considering a longest path and a disjoint path. Then, a job-shop scheduling

problem is solved where pairs of jobs that are in a resource conflict also need to be scheduled disjunctively in the job-shop instance.

Node packing bound Mingozzi et al. [186] propose to relax and reformulate the standard IP formulation for RCPSP. A weighted node packing problem remains that is still $\mathcal{NP}$-hard. They introduce variables which indicate that a (resource-feasible) subset of jobs is in process at a certain time t. This formulation is clearly of exponential size. They derive other lower bounds that are faster to compute. Their lower bound LB_3 identifies a clique of disjunctive jobs. Then, the sum of the processing times of these jobs leads to a lower bound. Several procedures to identify these cliques and further extensions have been developed, see e.g., [84, 159, 186].

Brucker and Knust [37] extend the formulation by Mingozzi et al. [186]. In their formulation, resource-feasible subsets of jobs must be scheduled preemptively in time intervals. Still, the number of variables is exponential and they propose a column generation approach. With this refinement they obtained the best known bounds on the majority of instances from PSPLib [162]. However, their algorithm requires very high running times. Baptiste and Demassey [19] use this formulation together with new valid inequalities in order to derive tight lower bounds, still with high computational efforts successful on small sized instances but not on the large ones.

Lower bounds from $Pm|\mathsf{r}_j, \mathsf{d_j}|C_{\max}$ [45]. For machine scheduling problems[1] several priority rules exist that often optimally or approximately solve the problem. In Jackson's rule [149], the jobs are sorted in non-decreasing order of their due dates. Then, maximum lateness is minimized for $1|\mathsf{d}_j|\max(C_j - \mathsf{d}_j)^+$. This rule and variants thereof have been used algorithmically for makespan minimization problems. Though $1|\mathsf{r}_j, \mathsf{d}_j|C_{\max}$ is strongly $\mathcal{NP}$-hard [117], its preemptive version $1|\mathsf{r}_j, \mathsf{d}_j, \mathsf{pmtn}\,|C_{\max}$ can be solved in $O(n \log n)$ by scheduling among all available jobs the one with minimum remaining latest start time and possibly preempting a job when a new release date is reached and another job has minimum remaining latest start time. It has been shown that the minimum makespan of the preemptive and non-preemptive schedule differ by at most $p_{\max} := \max_j\{p_j\}$ [41, 45].

The preemptive generalization to m machines $Pm|\mathsf{r}_j, \mathsf{d}_j, \mathsf{pmtn}\,|C_{\max}$ can be solved in $O(n^3(\log n + \log p_{\max}))$ [145]. This time complexity forbids its usage throughout a branch-and-bound algorithm. A further relaxation leads to a pseudo-preemptive schedule where a job can be executed by a rational number of machines and maybe more than one machine at once. This schedule is called *Jackson's pseudo-preemptive schedule (JPPS)* by Carlier and Pinson [45] who propose an $O(n^2 + nm^2)$ algorithm for JPPS and an $O(n \log n + nm \log m)$ algorithm for computing the optimum makespan of this pseudo-preemptive schedule. It has been confirmed that the optimality gap compared to an optimal non-preemptive schedule is at most $2 \cdot p_{\max}$. This property makes its use interesting throughout branch-and-bound.

Furthermore, Carlier and Pinson [45] present propagation algorithms for $Pm|\mathsf{r}_j, \mathsf{d}_j|C_{\max}$ and for the `cumulative` constraint that make explicit use of this rule and run in $O(n^2)$. Klein and Scholl [159] derive a similar bound by creating an m-machine relaxation where they introduce several copies of a job. They give an $O(n^3)$-algorithm. We remark here that energetic reasoning [22] is an extension of

[1] Recall the notation for machine scheduling problems from Section 1.2.

Algorithm 5: Algorithm to compute the preemptive lower bound $\text{LB}_{\mathsf{pmtn}}$ for one cumulative constraint.

Input: Resource capacity R, set $\mathcal{J}$ of jobs.
Output: Lower bound on the makespan $\text{LB}_{\mathsf{pmtn}}$.

1. Sort events $\{\text{est}_j, \text{ect}_j\}_j$, denoted by $e[]$.
2. $t_2 := 0$, $D := 0$, $t = 0$, $V = 0$.
3. **while** $t < n$ **do**
4. Set $t_1 := t_2$.
5. $t_2 :=$ next event $e[]$.
6. Set $V := \max\{0, V + (D - R) \cdot (t_2 - t_1)\}$.
7. Add demand of jobs that start at t to D.
8. Subtract demand of jobs that end at t from D.
9. **if** $t_2 + \lceil V/R \rceil > \text{ub}(C_{\max})$ **then**
10. **return** infeasible.
11. **if** $V > 0$ **then**
12. $\text{LB}_{\mathsf{pmtn}} := \max\{\text{LB}_{\mathsf{pmtn}}, t_2 + \lceil V/R \rceil\}$.

their rule to the cumulative scheduling problem, but it is less efficient. Second, we note that we use a similar technique in preprocessing in Section 3.2.3 where distances between pairs of jobs are updated globally by computing a preemptive lower bound. Our algorithm runs in $O(n \log n)$ for a particular pair of jobs given all the jobs in between. Hence, we do not use the algorithms proposed by Carlier and Pinson.

3.1.2 A new preemptive lower bound in $O(n \log n)$

We propose an efficient lower bound that combines the knowledge from precedence and resource constraints. This algorithm is sketched in Algorithm 5 and works as follows. We create an earliest start schedule according to the local lower bounds. This schedule may violate the capacity of some constraints. Next, we consider each `cumulative` constraint separately. Its resource-profile which has at most $2n$ changes can be built in $O(n \log n)$ from the earliest start schedule. Then, we sequentially scan the profile from left to right. During this scan, we collect the work that exceeds the capacity and shift this work into the holes not earlier than before where the capacity is not exceeded. If the end of the earliest start schedule is reached, all remaining work is divided by the capacity and rounded up to the next integer value. This way, we obtain a lower bound on the makespan. If this lower bound is larger than a given upper bound on the makespan, an infeasibility is detected. Symmetrically, we use a latest start schedule and run through the profile from right to left in order to shift the earliest possible start time of the project.

We denote this lower bound by $\text{LB}_{\mathsf{pmtn}}$.

Explaining bound updates of the preemptive propagator In order to use conflict analysis efficiently throughout search, any derived bound change must be explained. In case of the preemptive lower bound, a set of jobs with their accumulated demand V is responsible for updates of the lower bound of the makespan variable.

Finding this set of jobs can be easily done by calculating the last point in time when the value V has been set larger than zero. This point in time can be easily tracked during

Algorithm 6: Algorithm to explain a propagation by the preemptive lower bound $\mathrm{LB}_{\mathsf{pmtn}}$.

Input: Resource capacity R, set $\mathcal{J}$ of jobs, new bound b to be explained.
Output: An explanation $J \subseteq \mathcal{J}$ for $\mathrm{LB}_{\mathsf{pmtn}} = b$.

1 Set $V := (b - \mathrm{ect}_{\mathcal{J}} - 1) \cdot R + 1$.
2 Sort event points $\{\mathrm{est}_j, \mathrm{ect}_j\}_j$, denoted by $e[]$.
3 $t_2 := \mathrm{ect}_{\mathcal{J}}$, $D := 0$.
4 **while** $V > 0$ **do**
5   Set t to next smaller event in $e[]$.
6   Add demand of jobs that start at t to D.
7   Subtract demand of jobs that end at t from D.
8   Set $V := \max\{0, V - (D - R) \cdot (t_2 - t)\}$.
9   Set $t_2 := t$.
10 // Now, t_2 is the point in time for which the update can be derived.
11 Sort all jobs j with $\mathrm{ect}_j > t_2$ by $\min\{r_j \cdot p_j, r_j \cdot (\mathrm{ect}_j - t_2)\}$.
12 Collect $J \subseteq \mathcal{J}$ jobs in that order until
$t_2 + \left\lceil \sum_{j \in J} r_j \cdot \min\{p_j, \mathrm{ect}_j - t_2\}/R \right\rceil \geq \mathrm{LB}_{\mathsf{pmtn}}$.
13 **return** J.

the execution of Algorithm 5 in line 6. As we are using backward-checking and do not store that point in time with the bound change, we give an algorithm to compute the explanation. Furthermore, there may exist a smaller set of jobs that induces the same update or an update large enough to detect that the problem is infeasible. To find such a set of jobs, we execute the algorithm from right to left and track how much demand or volume of the jobs is still needed in order to derive this update. As soon as enough demand has been collected, we stop and report the corresponding set of jobs. Algorithm 6 formalizes these ideas. Recall that $\mathrm{ect}_{\mathcal{J}} := \max_j\{\mathrm{ect}_j\}$. Similar to explaining energetic reasoning propagation, we use in lines 11 and 12 that the set J of jobs must contribute enough energy such that the new bound $\mathrm{LB}_{\mathsf{pmtn}}$ is reached, i.e., it must hold that:

$$t_2 + \left\lceil \sum_{j \in J} r_j \cdot \min\{p_j, \mathrm{ect}_j - t_2\}/R \right\rceil \geq \mathrm{LB}_{\mathsf{pmtn}} .$$

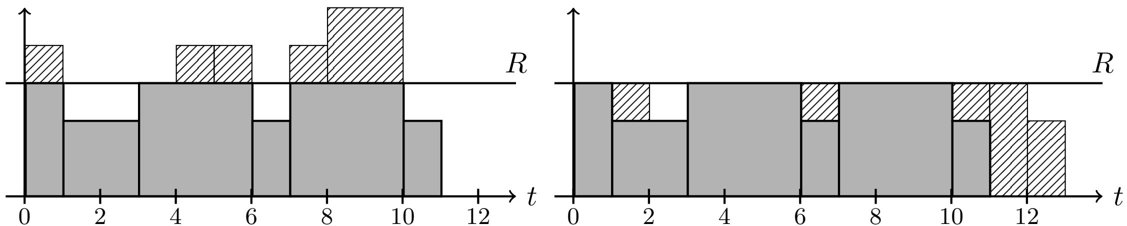


Figure 3.1: On the left, a profile of an earliest start schedule and on the right the shifted profile of all peaks that exceed the capacity are shown. The lower bound of the makespan increases from 11 to 13.

Preemptive lower bound vs. energetic reasoning The preemptive propagator is stronger than the edge-finding procedure, as this would only consider jobs that are fully contained in an interval $[a, b)$. The bound cannot be established by energetic reasoning, as in energetic reasoning (see Section 2.1.3.2), the energy contribution of a job j is given by

$$e_j^{\text{ER}}(a, b) = \max\{0, \min\{b - a, p_j, \text{ect}_j - a, b - \text{lst}_j\}\} \cdot r_j.$$

In the preemptive propagator, we assume that all jobs start at their earliest possible start time and that the makespan $C_{\max}$ is also shifted to the left immediately after the completion of all jobs. By that assumption, the energy contribution of a job is calculated assuming that each job is scheduled as early as possible. Thus, all demand that exceeds the capacity is shifted to the right.

$$e_j^{\mathsf{pmtn}}(a, b) = \max\{0, \min\{b - a, p_j, \text{ect}_j - a, b - \text{est}_j\}\} \cdot r_j.$$

Hence, our preemptive propagator uses the same arguments as energetic reasoning, but is restricted to shift the lower bound of the makespan variable. On the positive side, it runs in $O(n \log(n))$ instead of $O(n^2)$, the running time of energetic reasoning for one variable and all necessary intervals. On the one hand, we also use intervals that energetic reasoning could not consider with the same amount of energy, on the other hand, energetic reasoning spans a much larger range of intervals and is able to update all variables.

Impact on the solving process Initial experiments showed that this propagator does not cut off many nodes and if it does, the explanation consists of too many jobs as that reusable conflicts can be generated. This propagator reveals its strength as a preprocessing routine used on sub-networks. On these sub-networks, it is able to further improve the coefficients of transitive relations as we will see in Section 3.2.3. There, on average 17 transitive precedence relations can be strengthened on 660 out of 2,095 instances.

3.2 Presolving techniques

Prior to branch-and-bound search, most solvers perform a *presolving* in which simple or complex reduction techniques are used in order to fix variables, to tighten their bounds or to strengthen the constraints. The hope of its usage is that a much smaller problem with less redundant information remains to be solved. Often, inefficient but strong methods are applied here, since their usage throughout branch-and-bound does not overcome the high running times when used as propagation algorithms in every node.

In presolving, sometimes more than one constraint is considered at once. The propagation algorithms that have been presented in Section 2.1.3.2 concentrate on single `cumulative` constraints. It can certainly be expected that additional knowledge about a particular instance can be gained when precedence and resource constraints are considered together. In such an integrated presolving step, new constraints (stronger precedence relations or additional cumulative constraints) can be created.

We will introduce the following presolving techniques in the remainder of this section:

- Coefficient strengthening for cumulative constraints, see Section 3.2.1.

- Adding redundant resources with capacity one to the model, see Section 3.2.2.
- Stronger precedence constraints by volume arguments, preemption or disjunctive jobs, see Section 3.2.3.

3.2.1 Coefficient strengthening

Coefficient strengthening is a technique used in IP to strengthen the LP relaxation. Therefore, some coefficients in the linear inequalities are changed without changing the set of feasible solutions. E.g., inequality $23x_1 + 24x_2 + 25x_3 \leq 60$ can be strengthened to $x_1 + x_2 + x_3 \leq 2$ for $x_i \in \{0, 1\}, i = 1, 2, 3$. Then, the point $(1, 1, 13/25)$ is cut off.

Strengthening coefficients has proven very helpful in integer programming in order to yield better dual bounds. Andersen and Pochet [7] show that strengthening the coefficients sequentially proves useful in integer programming for production scheduling problems and on MIPLIB instances. They strengthen the Gomory cuts that are separated by CPLEX. Other authors considered constraints of the knapsack and subset sum type [89, 107] and report impressive results. We omit further references on this topic which spawns its own research direction. The advantage of strengthening coefficients is clearly that no additional inequalities or constraints are introduced such that the model is not overloaded with redundant constraints.

From a CP point of view it is not always clear what to gain from this technique. Strengthening the coefficients of a knapsack constraint would not have much impact as the following example shows.

Example 3.1. We consider a simple knapsack constraint:

$$\sum_j k \cdot x_j \leq K.$$

Here, each weight is equal to some integer value k. In case that $K = p \cdot k + (k - 1)$ for some integer p we know that in any feasible solution an amount of at least $k - 1$ will be left open. Hence, K can be decreased to $p \cdot k$. That would certainly 'only' give a better LP relaxation. In CP no further deductions could be made.

In case of scheduling, strengthened coefficients may result in stronger bound changes detected by propagation algorithms. In particular, energetic reasoning, edge-finding and time-table edge-finding or the preemptive lower bound use volume arguments to update the variable bounds or to detect infeasibilities. On the contrary, time-tabling cannot profit from this as no energy arguments are used. Hence, tightening the capacity of the resource or the resource demands of the jobs may improve the derived bound changes. This line of research may find further interesting applications in CP's constraints.

In the following, we consider a `cumulative` constraint: $\texttt{cumulative}(\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}, R)$. Let $\mathcal{J} = \{1, \ldots, n\}$ be the index set of the jobs watched by this constraint. For the cumulative constraint two kinds of coefficients can be tightened: the resource demands r_j and the capacity R. We now start from easy to more elaborate coefficient strengthening rules.

One of the easiest ways to update the resource demands works as follows. Compute the value $r^{\min} := \min\{r_j \mid j \in \mathcal{J} : 0 < r_j < R\}$ which is the minimum resource demand of all jobs j with non-zero demand smaller than the capacity. If any job has a resource demand that together with the minimum resource demand would exceed the capacity,

then it cannot be scheduled in parallel with any other job of this constraint and therefore its demand can be adjusted:

Corollary 3.1. *All resource demands* $r_j > R - r^{\min}$ *can be updated to* $r_j = R$.

Another simple coefficient strengthening technique can be applied by dividing all demands and the capacity by the greatest common divisor gcd.

Lemma 3.2. *Let* $d := \gcd(\{r_j \mid j \in \mathcal{J} : 0 < r_j < R\})$. *Then,* $r'_j := \lfloor r_j/d \rfloor$ *and* $R' := \lfloor R/d \rfloor$ *are valid coefficients for all* $j \in \mathcal{J}$.

Proof. Let S be a feasible solution of a cumulative constraint. Then, for each point in time t, $\sum_{j:S_j \leq t < S_j + p_j} r_j \leq R$ must hold. Division by d yields: $\sum_{j:S_j \leq t < S_j + p_j} r_j/d \leq R/d$. For the left part of the inequality is integer, we can round the right part to $\lfloor R/d \rfloor$. For infeasible schedules S we verify: $\sum_{j:S_j \leq t < S_j + p_j} r_j/d > R/d \geq \lfloor R/d \rfloor$. □

Besides these rather simple procedures, we establish next more sophisticated approaches. Using a knapsack routine, the resource capacity R can be adjusted to the maximum possible sum of demands less or equal to R. This can be accomplished by creating a knapsack instance with n items where the size and profit of item j correspond to r_j and the capacity is given by R. Again, only jobs with non-zero demand smaller than the capacity need to be considered. This approach never found an update in our experiments. Hence, we sharpened this idea.

A more elaborate approach to tighten the capacity R works as follows. If at any point in time t every feasible combination of jobs $J_t := \{j \in \mathcal{J} \mid \text{est}_j \leq t < \text{lct}_j \wedge 0 < r_j < R\}$ needs at most a capacity of R', then R can be reduced to R'. This can be checked for one point in time t by solving a knapsack problem for set J_t in which as above the capacity of the knapsack is set to R and the weights and profits of the knapsack problem are set to r_j. Observe that only values for t with $t = \text{est}_j$ for $j = 1, \ldots, n$ must be checked. We assume that a routine $\texttt{knapsackdp}(\boldsymbol{w}, \boldsymbol{p}, W)$ is available, that solves the maximum knapsack problem and returns the maximum value for a given set of items with weights $\boldsymbol{w}$, profits $\boldsymbol{p}$ and knapsack capacity of W in $O(n \cdot W)$. This coefficient strengthening technique is sketched in Algorithm 7 and runs in $O(n^2 \cdot R)$.

Lemma 3.3. *Given* $\texttt{cumulative}(\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}, R)$. *Algorithm 7 correctly strengthens the capacity in* $O(n^2 \cdot R)$.

Proof. Only points in times t with $t = \text{est}_j$ for $j = 1, \ldots, n$ must be checked since at any intermediate point the set of jobs is not increased. Each value needs to be checked once. We define $R_t := \max_{J \subseteq J_t} \{\sum_{j \in J} r_j \mid \sum_{j \in J} r_j \leq R\}$ which is computed for every point in time t in line 4 via a dynamic program for the knapsack problem. The tightest possible capacity that this algorithm can achieve is obtained through $R' := \max_t R_t$ in line 13.

It is easy to see that any infeasible combination J of jobs at some point in time t is still infeasible since the resource capacity has been decreased, hence, $\sum_{j \in J} r_j \geq R \implies \sum_{j \in J} r_j \geq R'$.

The fact that every feasible combination of jobs is still feasible is invoked by the algorithm by choosing R' as the maximum over all cumulated demands that may occur at any point in time. Especially, every resource demand $r_j = R$ is set to $r_j := R'$ afterwards in line 11.

Algorithm 7 needs $O(n \log(n))$ for the sorting step. There are at most n points in time to be considered and setting up one DP and solving it requires $O(n \cdot R)$ time. □

Algorithm 7: Coefficient strengthening for capacity R of `cumulative`$(\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}, R)$.

```
   Input: cumulative(S, p, r, R).
   Output: Strengthened capacity R' ≤ R.
 1 Set R' := 1.
 2 for t ∈ {est_j}_j do
 3 |  Compute J_t := {j ∈ 𝒥 | t ∈ [est_j, lct_j[}.
 4 |  R_t := knapsackdp(r_{J_t}, r_{J_t}, R).
 5 |  if R_t = R then
 6 |  |  return.
 7 |  end
 8 |  Set R' := max{R', R_t}.
 9 end
10 foreach job j ∈ 𝒥 with r_j = R do
11 |  Set r_j := R'.
12 end
13 Set R := R'.
```

In a similar fashion as the capacity is tightened in Algorithm 7, the demands can be tightened. In the following update, no temporal or precedence constraints are used.

Lemma 3.4. *Let* `cumulative`$(\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}, R)$ *be given with index set* $\mathcal{J} = \{1, \ldots, n\}$*. All resource demands* $r_1, \ldots, r_n$ *can be strengthened in* $O(n^2 \cdot (R - r^{\min}))$*.*

Proof. To perform this update we need to exclude in each step one job j and execute algorithm `knapsackdp` on all jobs that may be run in parallel to j. This can be done in $O(n \cdot (R - r_j)) \leq O(n \cdot (R - r^{\min}))$. We have n jobs, setting up the sub-instance and running `knapsackdp` takes $O(n^2 \cdot (R - r^{\min}))$. □

Observe that the coefficient strengthening technique of Lemma 3.4 must be performed sequentially. Otherwise, the resource demands may not be valid.

We now mention some further refinements. Somewhat stronger than Corollary 3.1 we can increase a resource demand r_j to R for some job j if no other job i can be executed in parallel to this job. Here, not only resource but also precedence constraints can be respected in order to restrict the set of jobs. Observe that only pairs i, j with $[\mathrm{est}_i, \mathrm{lct}_i) \cap [\mathrm{est}_j, \mathrm{lct}_j) \neq \emptyset$ and such that i is no predecessor of j according to the whole precedence graph need to be respected. Algorithm 8 states an algorithm to perform these coefficient strengthenings.

Lemma 3.5. *Let* `cumulative`$(\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}, R)$ *be given with index set* $\mathcal{J} = \{1, \ldots, n\}$*. All resource demands* $r_1, \ldots, r_n$ *can be strengthened in* $O(nm + n^2 \cdot (R - r^{\min})))$ *with respect to precedence and temporal constraints.*

We omit the correctness proof here. Line 6 of Algorithm 8 can be implemented in $O(m + n)$ by using a BFS in the precedence network.

Again, the resource demands must be strengthened sequentially. Hence, the order in which the jobs are considered plays an important role. Maybe, a whole optimization problem wants to be solved in order to strengthen all resource demands at once and as good as possible with respect to some measure. Nevertheless, we must be careful with

Algorithm 8: Coefficient strengthening for resource demands r_j of `cumulative`$(\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}, R)$ using time intervals and precedence relations.

Input: `cumulative`$(\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}, R)$.
Output: Strengthened resource demands r_j.
1 **foreach** *job* $j \in \mathcal{J}$ **do**
2   **if** $r_j = 0$ *OR* $r_j = R$ **then**
3   continue.
4   **end**
5   Compute $J_j := \{i \in \mathcal{J} \setminus \{j\} \mid [\text{est}_i, \text{lct}_i) \cap [\text{est}_j, \text{lct}_j) \neq \emptyset\}$.
6   Erase from J_j all jobs i that are precedence related to j.
7   Set $R' :=$ `knapsackdp`$(\boldsymbol{r}_{J_j}, \boldsymbol{r}_{J_j}, R - r_j)$.
8   Set $r_j := R - R'$.
9 **end**

the updates in order to get good ones. For example, a job may be updated that will not be involved in propagation or explanations, while another job cannot be updated any more. Intuitively, it is better to first reduce the capacity and then select jobs to be updated, as this fairly distributes the adjustments over all jobs. To this end, we execute these coefficient strengthening techniques in the following order: Corollary 3.1, Lemma 3.2, Lemma 3.3 and last Lemma 3.5. By using Corollary 3.1 first, we set the demands of very large jobs equal to the capacity. These won't be considered in the latter strengthening approaches. Then, we compute the greatest common divisor of all jobs with smaller resource demand than the capacity, see Lemma 3.2. Next, we strengthen the capacity according to Lemma 3.3 and last, we strengthen the single demands in an arbitrary order as indicated by Lemma 3.5.

We show that by these simple but highly efficient coefficient strengthening techniques, 20% more of the Pack instances can be solved to optimality if propagation algorithms such as edge-finding, time-table edge-finding or energetic reasoning are used. The strengthened coefficients lead to stronger volume arguments. The gap in the root node and the final gap can be decreased by 3% using this technique.

Related work Similar ideas have been proposed by Carlier and Néron [43] who compute maximum redundant functions (MRFs) that induce higher volume ratios. A redundant function f maps $\{0, \ldots, R\}$ to $\{0, \ldots, R'\}$. The function f is applied to all demands and the capacity of one cumulative constraint while preserving resource feasibility. The authors show that only maximal redundant functions are of interest and that their number is low for $R \leq 10$ (at most 48 such maximal redundant functions) but high already for $R = 20$ where 393 such functions exist.

3.2.2 Redundant resources

In many benchmark instances, jobs are present with resource demands larger than half the capacity. These are pairwisely not allowed to be executed in parallel. A `disjunctive` constraint can be created for such pairs, see Section 2.1.2. Even better, if a whole clique of jobs can be detected that cannot be executed in parallel, then a cumulative constraint with capacity one can be introduced. Thus, volume arguments like in edge-finding or

Algorithm 9: Building redundant resources.

Input: Set $\mathcal{J}$ of jobs and their disjunctive graph, a minimum cardinality c and minimum load factor L.
Output: Additional cumulative constraints of capacity one.

```
1  foreach job j sorted by non-decreasing lct_j do
2      Set S := {j}.
3      foreach job i sorted by non-increasing lct_i do
4          if lct_i ≤ lct_j then
5              continue.
6          end
7          if i disjunctive with j OR i ≺ j then
8              foreach k ∈ S do
9                  if k is not disjunctive with i and (i ⊀ j and j ⊀ i) then
10                     continue.
11                 end
12             end
13             Set S := S ∪ {i}.
14         end
15     end
16     if |S| ≥ c and p(S) ≥ L · (lct_S − est_S) then
17         Store S.
18     end
19 end
20 Remove redundant and irrelevant sets S from storage.
21 Create cumulative constraints for each set S from storage.
```

energetic reasoning can be applied in order to fasten the search and to detect stronger bound changes, see e.g. Carlier and Néron [44].

Building such redundant cumulative constraints with capacity one can be done constraint-wise or by taking into account all disjunctions that are implicitly given by the set of all cumulative constraints. Recall that jobs need several resources at once and may therefore not be allowed to be executed in parallel according to different `cumulative` constraints. Additionally, precedence constraints can be taken into account in order to increase the size of the set of jobs in disjunction and also the domains of the variables can be used. If two domains do not overlap, the jobs are disjunctive.

In Algorithm 9 we sketch how the redundant resources of capacity one are build. We sort the jobs by non-decreasing latest completion time and set up for each job j a large disjunctive clique S greedily. A job i is added to S if it is in disjunction with all elements so far added to S.

In our experiments, we keep at most the ten largest such sets S with at least $c = 6$ variables and only if the sum of the processing times is at least $L = 70\%$ of the total interval length induced by these jobs. Note, that $\text{lct}_S = \max_{j \in S}\{\text{lct}_j\}$ and $\text{est}_S = \min_{j \in S}\{\text{est}_j\}$.

Redundant resources have already been used in other studies [19, 21, 44]. In particular, Baptiste and Demassey [19] propose to solve MIPs and build many redundant resources that maximize the sum of the processing times on the redundant resource.

They build one redundant constraint for each cumulative constraint and one for all constraints. In contrast, we use the disjunctive cliques to build redundant resources as it turned out that redundant constraints from a single cumulative resource constraint yield not much improvement after the propagation algorithms of the cumulative constraints are executed.

3.2.3 Strengthening transitive precedence constraints

In this section we consider precedence and resource constraints simultaneously in order to derive precedence relations that are beyond longest distance computations in the precedence graph.

With the following presolving techniques, we tighten the minimum distance between the start times of two jobs. We consider pairs of jobs that are in a transitive precedence relation. Hence, their order is given and a longest path from the predecessor to the successor yields the minimum distance between these two jobs that must be respected in every feasible schedule. This minimum distance can be strengthened by considering the set of jobs that must be scheduled between these two job. In particular, we will use knowledge about disjunctive relations and volume arguments to strengthen the minimum distance.

The pair of jobs is denoted by (i, j), with $i, j \in \mathcal{J}$ and $i \neq j$. We denote by J_{ij} the set of all jobs k that succeed job i ($k \in \mathit{Succ}(i)$) and precede job j ($k \in \mathit{Pred}(j)$) in the precedence graph. Formally, we write $J_{ij} := \{k \in \mathcal{J} : k \in \mathit{Succ}(i) \wedge k \in \mathit{Pred}(j)\}$.

All jobs $k \in J_{ij}$ must be scheduled before the start of job j and after job i has been completed. Hence, a schedule of minimum length (or makespan) for the subset of jobs yields a valid lower bound on the minimum distance between jobs i and j in any feasible schedule. Because solving this problem is hard, bounds are sought after. In the following we derive lower bounds by (i) a simple volume argument, (ii) a preemptive schedule and (iii) by disjunctive jobs.

For simplicity we consider one cumulative constraint and omit the index of this resource in this section.

Distance by volume arguments The *energy* (or work volume) of set J_{ij} is given by

$$E(J_{ij}) := \sum_{k \in J_{ij}} r_k \cdot p_k$$

A lower bound on the completion time of the subset J_{ij} is obtained by dividing the energy by the capacity as summarized in the following lemma.

Lemma 3.6. *In any feasible schedule, the minimum distance between jobs i and j is at least $d_{ij} = E(J_{i,j})/R$. Hence, $S_j \geq S_i + p_i + d_{ij}$ holds.*

Observe that this bound can be computed in linear time $O(n)$ given the set of jobs J_{ij}.

Preemptive distance Computationally more expensive but very often much stronger, the distance can be updated with respect to a preemptive schedule.

Lemma 3.7. *In any feasible schedule, the makespan of a preemptive schedule for the set of jobs J_{ij} is a valid bound on d_{ij}. Hence, $S_j \geq S_i + p_i + \mathrm{LB}_{\mathsf{pmtn}}(J_{ij})$ holds.*

We compute this preemptive schedule by computing LB_{pmtn} for the set of jobs J_{ij} as already done in Section 3.1.2. We point out that the earliest start times must not be based on the global bounds of the variables as in general jobs i and j can be slided in their time windows. Hence, we must recompute the earliest start times for each set J_{ij} according to the precedence constraints to obtain a valid bound.

Additionally, we reverse the precedence structure and compute the preemptive schedule for that order. Then, the maximum over both orders yields a valid lower bound on the distance d_{ij} between the completion of job i and the start of job j.

Distance by disjunctive relations Let $k_1, k_2 \in J_{ij}$ such that k_1 is in disjunction with k_2. Then, by two simple longest path calculations, a lower bound on the makespan for the set of jobs in J_{ij} can be computed. Therefore, the disjunction is temporarily resolved by imposing once each precedence relation and computing the length of a critical path in that precedence graph. The minimum of both makespans yields a valid lower bound on d_{ij} as summarized in the next lemma.

Lemma 3.8. *Let C^1 be the length of the longest path in the precedence graph for the set J_{ij} where* `precedence`$(S_{k_1}, S_{k_2}, p_{k_1})$ *is imposed and let C^2 be the length of the longest path in the precedence graph for the set J_{ij} where* `precedence`$(S_{k_2}, S_{k_1}, p_{k_2})$ *is imposed.*

Then, $\min\{C^1, C^2\}$ *is a valid lower bound on the minimum makespan of J_{ij}.*

This way a minimum distance between jobs i and j is only improved if in an earliest and in a latest start schedule the jobs k_1 and k_2 are scheduled in parallel. Then, both longest paths are increased with the imposed precedence constraints.

Figure 3.2 depicts a typical network, as contained in instances from PSPLib, with disjunctive relations between jobs i_2 and i_3 by which the distance between jobs i_1 and i_4 can be updated. Several sub-networks, such as $J_{S_0 i_{10}} := \{i_1, i_4, i_6, i_7, i_8\}$, can be found in the graph to which volume arguments can be applied. For the pair (i_6, i_{11}), a disjunctive bound between jobs i_9 and i_{10} might be of interest as well as volume arguments or the preemptive bound of all jobs between i_6 and i_{11}. Between i_4 and $C_{\max}$ depending on the processing times of jobs i_4, i_5, i_{10} and i_{11} transitive relations due to disjunctions may be obtained.

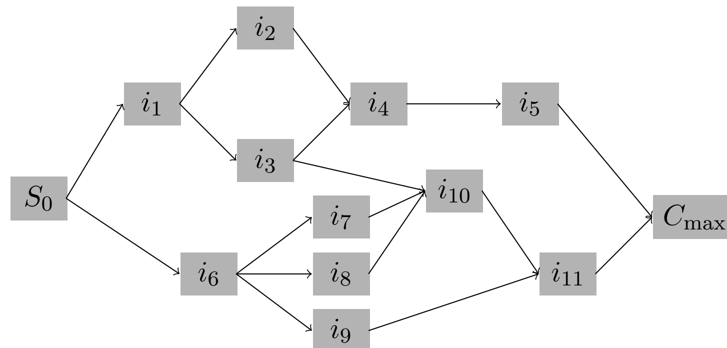


Figure 3.2: A precedence network indicating potential sub-networks for transitive precedence relations.

3.2.4 Computational study

In the preceding section, we recapitulated lower bounds on the makespan and adding redundant resources to the model. Furthermore, we developed a preemptive propagator to derive lower bounds on the makespan and we derived sufficient conditions under which the demand coefficients and the capacity can be strengthened per cumulative constraint without adding any redundant constraints. We proposed to detect sub-networks on which these lower bounds can be applied.

With the following study we evaluate the impact of these procedures on the solving process. In particular, we present how often the single presolving steps occur and by how much the dual bounds in the root and after solving can be improved this way.

3.2.4.1 Occurrences of presolving techniques

Table 3.1 shows the number of occurrences of the different presolving techniques on all instances from PSPLib and Pack which together are 2,095 instances. The first row 'all' shows the results on all instances while the second row 'min' shows the results on all instances that are not solved to optimality after presolving using the proposed techniques.

There are only few instances on which the coefficient strengthening techniques occur, here, most often capacity and demands can be normalized. On about half of the instances we find jobs that are pairwise in disjunction. Among the PSPLib instances with 120 jobs, there is an instance that even contains 2619 disjunctive pairs, while the average value is 193. Only on 109 instances we are able to find constraints of capacity one that contain at least 10 variables or fill up the space between the earliest start and latest completion time by at least 70%. On more than half of the instances we detect sub-networks, on average 60 per instance. Using these, we are able to tighten transitive precedence relations according to volume arguments on 577 instances, according to disjunctions on 618 instances and to further improve the distance on a transitive edge via the preemptive propagator on 660 instances. We point out that on each sub-network we first compute the bound by volume arguments, then by disjunctions and finally by the preemptive propagator, hence there may be some more counters for the preemptive propagator where it cannot improve the bounds derived from the other techniques.

Using all presolving techniques, we are able solve 1,164 of all 2,095 instances in the root node. This is only a slight improvement of 21 instances, as without presolving already 1,143 instances are solved in the root node. This does not yet indicate any huge gain when performing exhaustive presolving. We will see that the impact on the overall solving process is much higher.

3.2.4.2 Impact of coefficient strengthening

As the time-tabling propagation algorithm does not use volume arguments, it does not benefit from strengthened coefficients. In contrast, the energy based propagation algorithms, such as edge-finding 'coefEF', time-table edge-finding 'coefTTEF' and energetic reasoning 'coefER', and the binary relaxation 'coefBinvar' may potentially take advantage of this technique. Settings, where the presolving steps are used, are marked by a 'w'. We run all these algorithms in combination with time-tabling on the Pack instances and on PSPLib. On the latter, not much effects can be seen, hence, we omit the results.

The influence of coefficient strengthening procedures besides normalization is hard to measure, as most of the given instances are easy to solve except for 4 instances from

Table 3.1: Occurrences of the presolving techniques on all instances. Column 'total' gives the number of instances where the presolving technique applied, 'max' the maximum number of occurrences per instance and 'av.' and 'dev.' the arithmetic mean and standard deviation of those instances where the technique applied.

	coef strengthening				cap tightening				normalize				disjoint				cum1 by disj			
	total	max	av.	dev.	total	max	av.	dev.	total	max	av.	dev.	total	max	av.	dev.	total	max	av.	dev.
all	134	11	2.3	1.8	34	3	1.3	0.6	35	72	14.1	15.4	957	2619	193.0	347.8	109	5	1.5	0.8
min	57	7	2.4	1.7	9	3	1.6	0.7	5	36	20.2	9.7	532	2619	319.5	423.9	97	5	1.5	0.8

	sub-networks				prec by volume				prec by disjunction				prec by pmtn			
	total	max	av.	dev.	total	max	av.	dev.	total	max	av.	dev.	total	max	av.	dev.
all	1129	100	60.7	35.9	577	100	14.7	22.4	618	50	9.6	8.9	660	78	16.9	17.5
min	930	100	68.1	33.4	543	100	15.6	22.8	582	50	9.8	9.0	623	78	17.6	17.7

Table 3.2: Impact of coefficient strengthening: This technique enables us to solve about ten more instances per propagation routine from the set Pack; where Pack045 is solved for the first time.

setting	nopt	bprimal	bdual	gapRootDual	gapRoot	gap	avtime	avnodes
Pack (55 instances)							(allopt: 15 instances)	
coefEFw	29	54	52	-0.42	7.02	3.28	6.87	25035.8
coefEF	19	53	41	-3.31	10.52	6.61	6.87	25035.8
coefTTEFw	29	54	55	-0.42	7.02	3.24	7.52	24247.8
coefTTEF	18	52	43	-3.31	10.52	6.59	7.52	24247.8
coefERw	25	50	52	-0.42	7.02	3.60	47.71	23372.8
coefER	15	49	40	-3.31	10.52	7.02	47.71	23372.8

set Pack. Hence, in Table 3.2 all instances that have been solved to optimality with and without coefficient strengthening do not differ enough to show an impact. On the one hand, the success of this technique is supported by about ten more instances that can be solved additionally by whatever propagation algorithm is used. On the other hand, we also see that using strengthened coefficients decreases the gap in the root node from 10.52% to 7.02%. This gain is due to the better dual bounds and not due to better primal solutions, as the dual bound in the root node decreases from 3.31% to 0.41% (as column 'gapRootDual' reveals). Finally, this gives an overall improvement on the final gaps by about 3% if coefficient strengthening is applied due to generally better dual and few (one or two) better primal bounds.

Considering the binary relaxation, we not only strengthen the coefficients of the cumulative constraint, but we also strengthen the capacity cuts per point in time. Such a strengthening occurs at least once per instance.

The results as shown in Table 3.3 are weird at a first glance. On the Pack instances using coefficient strengthening, we solve two instances less (Pack 23, Pack 24) than before, though this procedure does not need much running time. A closer look reveals that the final dual bound on these instances is optimal and that the optimal solution is not found though about 10 times more nodes are explored within the remaining time limit. Hence, this outcome must be due to different branching decisions. Second, using strengthened coefficients the LP solutions may get more degenerated which is a reason why the LP solving times on several instances (such as the two above) increase by about 25% per node whereas the propagation times remain the same. On set setL, the average number of

Table 3.3: Impact of coefficient strengthening on the binary relaxation.

setting	nopt	bprimal	bdual	gapRootDual	gapRoot	gap	avtime	avnodes
setS (114 instances)							(allopt: 89 instances)	
coefBinvar-w	91	114	109	-18.55	37.74	2.17	51.57	6975.99
coefBinvar	89	112	107	-18.60	37.85	2.35	52.92	7149.04
setL (119 instances)							(allopt: 90 instances)	
coefBinvar-w	93	117	113	-7.34	18.63	2.22	64.59	3176.99
coefBinvar	92	116	111	-7.36	18.71	2.33	67.54	2781.28
Pack (55 instances)							(allopt: 36 instances)	
coefBinvar-w	36	53	55	-0.72	7.02	2.41	31.1	7050.08
coefBinvar	38	55	55	-0.76	7.06	2.47	31.1	7050.08

nodes even increases by more than 10% whereas the running time decreases slightly. On setS and setL, we see that strengthening the coefficients does not make much difference, only on two more instances a better primal or dual bound can be found. Neither the average final gap nor the dual gap in the root node decrease noticeably.

3.2.4.3 Impact of the presolving techniques

Table 3.4: Impact of presolving techniques.

setting	nopt	bprimal	bdual	gapRootDual	gapRoot	gap	avtime	avnodes
setS (114 instances)							(allopt: 98 instances)	
nopresol	99	111	109	-19.88	39.77	1.03	32.46	43081.11
subnets	101	113	107	-19.07	38.21	0.95	28.98	41597.62
disj	100	112	108	-19.88	39.77	0.98	34.15	39773.10
subdisj	101	113	109	-19.07	38.21	0.91	**25.88**	**32604.19**
subcum1	99	111	102	-19.07	38.21	1.27	51.51	41597.62
all	**102**	**114**	**111**	**-17.95**	**36.22**	**0.84**	30.58	35286.18
setL (119 instances)							allopt: 101 instances)	
nopresol	105	114	117	-8.32	19.55	0.66	27.12	25177.59
subnets	107	116	116	-8.01	19.06	0.50	**22.24**	21094.31
disj	107	116	117	-8.25	19.48	0.51	26.54	24095.87
subdisj	107	116	118	-7.94	18.97	0.49	24.57	22108.62
subcum1	105	114	115	-8.01	19.06	0.63	36.18	**21059.46**
all	**108**	**117**	**119**	**-7.78**	**18.73**	**0.43**	24.67	21406.66
Pack (55 instances)							allopt: 27 instances)	
nopresol	27	51	46	-1.59	7.06	3.38	3.58	6650.37
subnets	28	52	47	-1.54	7.02	3.31	3.61	7727.26
disj	28	52	47	-1.59	7.06	3.31	3.04	5269.48
subdisj	28	52	47	-1.54	7.02	3.31	3.48	6758.67
subcum1	28	52	46	-1.54	7.02	3.38	6.33	7727.26
all	**36**	**55**	**54**	**-0.44**	5.67	**1.86**	**2.31**	**4058.15**

We consider the following settings. Initially, all presolving techniques are turned off in setting 'nopresol'. Then, we run our CP-SAT framework with strengthened transitive precedence relations on the sub-networks, denoted by 'subnets', or with additional `disjunctive` constraints, denoted by 'disj'. Redundant resources that suffice our criteria are only found if transitive precedence relations are strengthened on the sub-networks. Combination of these settings are given by the settings 'subdisj' (combination of 'subnets' and 'disj'), 'subcum1' (a combination of 'subnets' and redundant resources) and

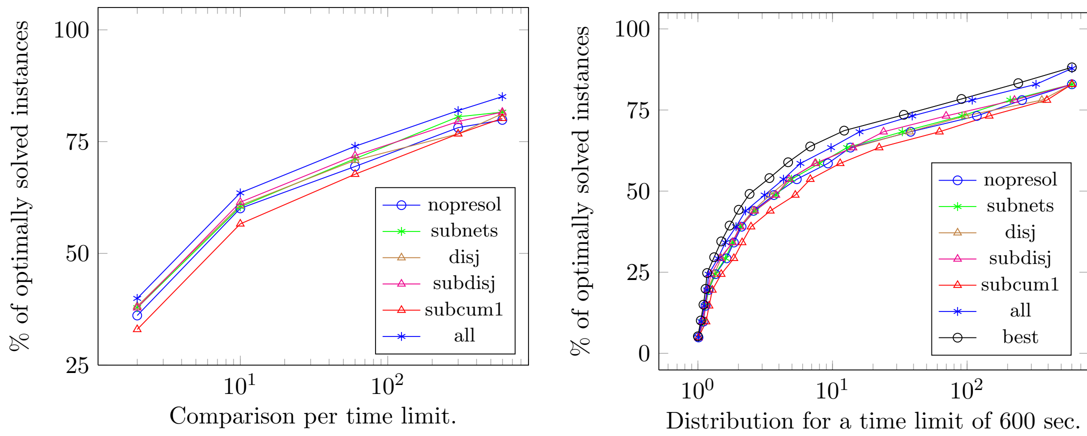


Figure 3.3: Percentage of optimally solved instances within different time limits for the proposed presolving techniques on all instances from setS, setL and Pack.

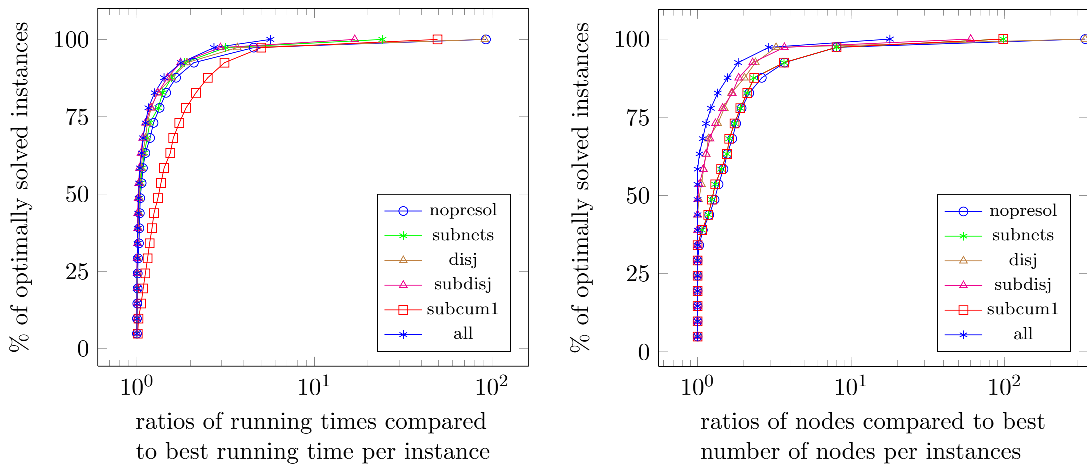


Figure 3.4: Comparison of nodes and running times for the different presolving techniques on all instances from setS, setL and Pack.

'all' where all presolving techniques are enabled. Time-tabling is used as propagator.

Table 3.4 shows that using presolving on the PSPLib instances only decreases the gap in the root node slightly, usually when sub-networks are considered. On the negative side, computing improved distances on the transitive relations combined with the redundant resources, i.e., setting 'subcum1', even needs much more running time on average than without presolving. In this setting on set setS, the average running time on all optimally solved instances is twice as high as the best average running time obtained when combining sub-networks with `disjunctive` constraints. From an instance to instance comparison we see that almost twice as many conflicts are generated, which do not propagate much more. Here, several of these conflicts may be redundant, which is not checked in the SCIP framework. There are several runs in which using the presolving techniques increases the number of nodes needed, in particular if sub-networks are considered.

The highest improvements when considering the number of optimally solved in-

stances, best primal and dual bounds, and the lowest final gaps are achieved when all presolving techniques are turned on. The average number of nodes on those instances solved to optimality by all settings is decreased by 20%, while the average running time is only decreased by less than 10% in contrast to not applying any presolving technique.

Figures 3.3 and 3.4 show that using sub-networks and redundant resources, the average number of nodes cannot be decreased much. The running time distribution becomes even worse than without any presolving. Adding `disjunctive` constraints yields most of the reductions in the running time and number of nodes, while using all techniques together remarkably improves the node distribution function.

3.2.4.4 Impact of presolving techniques on the propagation algorithms

Table 3.5: Impact of presolving techniques on the propagation algorithms.

setting	nopt	bprimal	bdual	gapRootDual	gapRoot	gap	avtime	avnodes
setS (114 instances)							(allopt: 75 instance)	
ttpres	102	112	110	-17.98	36.22	0.84	3.47	4283.83
tt	99	109	109	-19.9	39.77	1.03	3.38	5116.89
efpres	101	111	109	-17.98	36.22	0.87	4.28	3618.41
ef	98	108	105	-19.9	39.77	1.21	5.44	5301.45
ttefpres	99	109	105	-17.98	36.22	1.13	6.84	3714.63
ttef	96	106	102	-19.9	39.77	1.47	9.21	5467.91
erpres	81	91	84	-17.98	36.22	3.33	94.22	3949.45
er	76	86	82	-19.9	39.77	4.22	126.19	5185.85
binvarpres	90	100	93	-17.6	35.55	2.44	32.15	2351.91
binvar	91	101	94	-19.5	39.07	2.21	24.41	3400.75
setL (119 instances)							allopt: 70 instances)	
ttpres	108	113	119	-7.78	18.73	0.43	2.19	1452.89
tt	105	110	117	-8.32	19.55	0.66	2.23	1686.45
efpres	103	108	116	-7.69	18.62	0.87	3.03	1405.77
ef	104	109	115	-8.32	19.55	0.71	3.33	1780.13
ttefpres	103	108	115	-7.69	18.62	0.83	3.78	1136.03
ttef	103	108	114	-8.32	19.55	0.79	4.52	1541.28
erpres	80	85	98	-7.69	18.62	2.44	57.69	1017.85
er	74	79	95	-8.31	19.54	3.02	80.64	1489.37
binvarpres	92	97	102	-7.06	17.74	2.01	50.81	1046.39
binvar	92	97	99	-7.72	18.7	2.29	41.51	1165.24
Pack (55 instances)							allopt: 27 instances)	
ttpres	36	46	49	-0.62	5.67	1.86	2.31	4058.15
tt	27	42	43	-1.77	7.06	3.38	3.58	6650.37
efpres	38	48	51	-0.62	5.67	1.65	3.29	7006.74
ef	28	43	43	-1.77	7.06	3.34	3.26	5629.22
ttefpres	40	50	51	-0.62	5.67	1.52	7.08	15847.41
ttef	31	46	45	-1.77	7.06	3.09	6.4	12503.22
erpres	38	48	49	-0.62	5.67	1.78	10.76	3717.96
er	30	45	43	-1.77	7.06	3.31	10.36	3527.78
binvarpres	41	51	55	-0.62	5.67	1.33	3.37	144.89
binvar	38	52	49	-1.77	7.06	2.47	8.33	2971.93

Finally, we show how the presolving techniques influence the search if different propagation algorithms are applied. We remark that using only time-tabling propagation is the best choice on the PSPLib instances, while on the Pack instances, time-table edge-finding or the binary relaxation are worth being used. The propagation algorithms time-tabling ('tt'), edge-finding ('ef'), time-table edge-finding ('ttef') and energetic reasoning ('er') and the use of the binary relaxation ('binvar') are evaluated. If presolving is applied, we

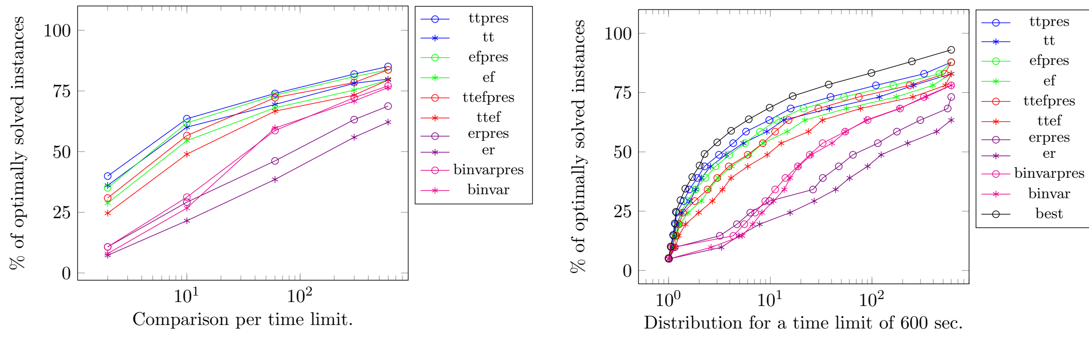


Figure 3.5: Number of optimally solved instances if problem-specific presolving is applied (marked by $\circ$) or not (marked by $\star$).

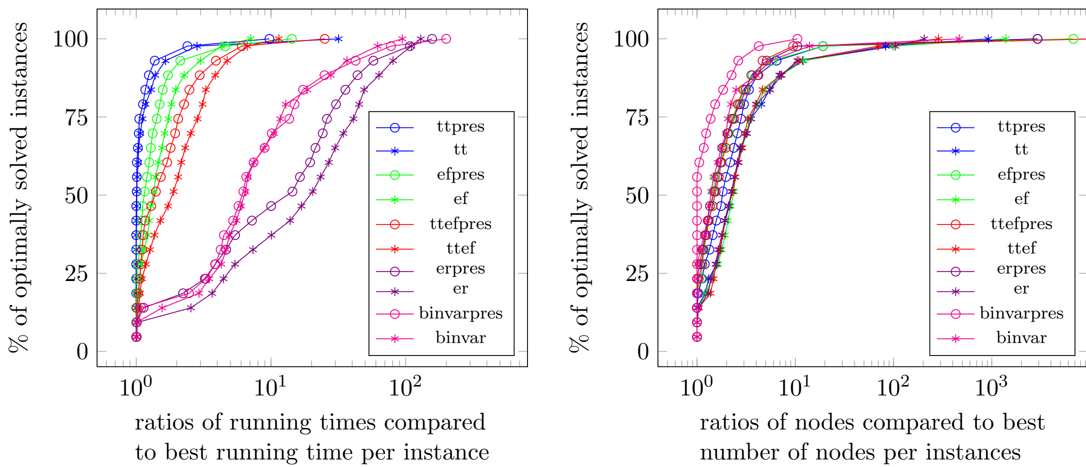


Figure 3.6: Ratios of nodes and running times given for all optimally solved instances per propagation algorithm with all presolving techniques turned on ('pres') and off.

append 'pres' to the settings.

Table 3.5 shows the results per instance set. Using presolving decreases the gap within the root node by 2% on setS and about 1% on setL and Pack. Considering the final gap, this is sometimes even increased, e.g., on setL in setting 'efpres' and 'ttef' due to higher propagation times. Considering those instances solved to optimality by all settings, the use of presolving results in 10 to 20% fewer nodes on average but only slight decreases in the running times on setS and setL. On the Pack instances, the average running time slightly increases for 'efpres', 'ttefpres' and 'erpres' and the average number of nodes increase by even 10%. Here, the propagation algorithms detect 10% more adjustments per node and 20% more conflicts are generated on average. The best solution is found much later which explains the increased number of nodes on some instances.

Considering the number of optimally solved instances and the number of best primal and best dual bounds, we see that these are usually increased between one and six instances if presolving is used. A few negative results remain. For 'efpres' we solve one instance less to optimality on setL, but ten more on Pack.

Using the binary relaxation combined with presolving, the average running times of

all optimally solved instances from PSPLib increase which is due to higher LP solving times and the additional time spent in presolving. On the Pack instances, the average running time decreases by one-half and the nodes to about 10% which is remarkable.

From Figures 3.5 and 3.6 we see that with respect to the running times and the number of optimally solved instances within certain time limits, the distribution functions are in favor of using the proposed presolving techniques. In total, we see that in particular on the highly cumulative Pack instances, using presolving is essential to improve the dual bounds.

Summary & conclusion

Generating optimal solutions for instances of RCPSP is challenging due to its hardness. In order to cope with this intriguing hardness, we propose to tighten the coefficients of the cumulative constraints which enabled us to solve some very hard instances from the Pack set already within few nodes. Furthermore, we decomposed the precedence network and identified sub-networks on which lower bounds based on volume arguments, due to disjunctive jobs and based on a new preemptive propagator can be derived.

Extending coefficient strengthening techniques from IP to CP's constraints potentially leads to further fruitful research in that area. In particular, the feasibility tests and propagation algorithms for machine constraints or minimum resource constraints may make use of such techniques.

3.3 Tree search algorithms

In this section we discuss different branching strategies that have been used in CP, IP and SAT solvers for scheduling problems. Some of these strategies are generic while others are problem specific. Furthermore, we conclude this chapter by comparing the CP, IP and SAT solving capabilities of our solver against the latest results from the literature.

In a branch-and-bound search, *branching* means to split the problem in two or more subproblems while the union of the feasibility spaces must contain all feasible solutions, or at least one optimal solution. This is done by creating child nodes and imposing new local constraints which can be lower and upper bounds on variables or new logical constraints. If n child nodes are created, we speak of *n-ary* branching, and in case $n = 2$ we speak of *binary branching*.

Branching decisions, such as the choice of the branching variable and its value (in case of non-binary variables) as well as the number of child nodes to be created, have a huge impact on the solving process. Hence, it is best to choose them wisely. Often, it is a *good* choice to create two subproblems that are of eager strength, e.g., the dual bound is eagerly increased or further propagations can be made in both subproblems.

For RCPSP several branching schemes have been developed throughout the literature. Knowledge, e.g., from earliest start schedules respecting the local bounds of the current node, can be used to perform branching on variables that are involved in resource conflicts. Similarly, new precedence constraints can be posted in a child node to resolve resource conflicts.

In contrast, generic branching schemes collect information throughout search in order to decide which variable is selected from the set of branching candidates. Prominent examples for such rules are given by scores based on (i) the improvement in the objective

function (IP), based on (ii) the number of bound adjustments (CP) or based on (iii) those variables that are often involved in conflict analysis (SAT).

Outline In Section 3.3.1, we start with a presentation of the most important scheduling specific branching schemes together with a discussion on branching on the makespan variable. Then, generic branching schemes are presented. In the computational study (Section 3.3.2) these strategies are compared with each other. Interestingly, on instances from PSPLib the generic branching schemes outperform the others, whereas on the highly cumulative Pack instances a scheduling specific branching scheme is able to solve more instances.

Finally, we compare our approach to the best results obtained in the literature. In particular, we evaluate the techniques from CP, IP and SAT in our framework SCIP by presenting the results when using a pure CP solver, a CP-SAT hybrid, with and without the continuous relaxation and in combination with the standard IP formulation.

Results The computational study shows that generic branching schemes which *learn* throughout search from former branching decisions work well on PSPLib instances. In particular, collecting vsids as done in SAT, i.e., a score how often a variable is involved in detecting bound adjustments turns out to be useful. This way more than 50% additional instances can be solved compared to the problem specific branching schemes.

In contrast, on the highly cumulative Pack instances, a problem specific branching scheme performs much better with respect to the number of optimally solved instances as well as the average running time and nodes. In this scheme, we branch on a variable that is involved in a resource conflict based on an earliest start schedule.

3.3.1 Branching schemes

3.3.1.1 Scheduling specific branching schemes

In a *schedule generation scheme search* (SGS search), we compute an earliest start schedule and the corresponding resource profiles for each cumulative constraint. Then, we find the first resource conflict and choose one of the conflicting variables. Branching on this variable is performed by fixing its start time to the earliest start time in one branch. And second, by fixing the variable to the minimum earliest completion time of all jobs that are in resource conflict with this variable. This branching scheme has been used by Baptiste et al. [21] and is also known as *time-oriented schedule generation scheme*, see Dorndorf et al. [93].

Liess and Michelon [177] use the following branching rule: Given a valid upper bound (e.g., from a heuristic) on the makespan $C_{\max}$, all jobs must finish before this upper bound minus one. Any feasible solution found throughout search can be used to further decrease the makespan. On the second level, an unfixed variable j is chosen that maximizes $\sum_k r_{jk} \cdot p_j$. In the first subproblem, $\{S_j \leq \text{est}_j\}$ is added and in the second subproblem $\{S_j \geq \text{est}_j + 1\}$ is added. They use a depth-first search strategy.

Schrage [223] uses the left-shift dominance-rule to guide the branching decisions. A job can be scheduled at its earliest start time if it does not create a resource conflict in an earliest start schedule. Branching is performed by scheduling a first undecided (not yet fixed) job as early as possible, or this job is delayed by the minimum of all earliest completion times of jobs that are executed in parallel to that job. Demeulemeester and

Herroelen [83] present a branching scheme that resolves conflicts by identifying the first conflicting set in an earliest start schedule. From the conflict set, *minimal delaying alternatives* are identified, i.e., all subsets of jobs that resolve the conflict when they are delayed at least at the considered point in time. These conflicts are then resolved by locally adding precedence relations to the subproblem for each minimal delaying alternative. They also use dominance rules that hold over these cut sets in order to prune unpromising nodes early.

Lombardi and Milano [179] consider a variant of RCPSP in which generalized precedence constraints are present and variables with processing times that are not known a priori but are specified in a range. They also post new precedence constraints during branching in order to resolve resource conflicts.

On instances of small size, such as the PSPLib instances containing 30 jobs, the dominance cut-set rule by Demeulemeester and Herroelen [84] is very efficient. In this rule, sets of jobs are stored whose predecessors have already been scheduled. Using these cut-sets, a node can be pruned if it contains the same cut-set as another node, otherwise, no better schedule for the same set can be found. This approach has been the first to close all instances from set J30, but is impractical for medium or large instances because of the high number of cut sets, as reported by many authors, e.g., [177].

Branching based on cores We propose another branching scheme that borrows ideas from time-tabling. In time-tabling, resource profiles of each cumulative resource are created that contain the cores of all jobs. Given some lower bounds on the start times or, if present, the start times in an LP solution, each job is scheduled as early as possible with respect to these profiles and the capacity value. In case a variable is shifted above its local upper bound, we select it as branching variable. In the first branch, we fix the variable on its upper bound, while in the second branch, its value is bounded by its upper bound minus one. Intuitively, we repair the earliest start schedule this way: A job cannot be scheduled due to its local time window and resource constraints. But it seems likely to be right-shifted. Hence, in the first branch, we fix the job to its latest start time and thereby create a new core that must be respected in the next branching round and may also trigger further propagation after the fixing.

Branching based on the cumulative constraint We combine the ideas from SGS and Liess and Michelon [177] in one scheme. First, we compute an earliest start schedule without resource constraints. Then, the resource profiles are scanned and the first resource conflict is taken. Among those variables that are part of the resource conflict, we select a variable with largest value $\sum_k r_{jk} \cdot p_j / R_k$. Hence, the chosen variable occurs in many constraints or has a high energy contribution in at least one cumulative constraint.

Branching on the makespan variable All branching schemes can be combined with a specified search that prefers branching on the makespan variable. Depending on the way in which the branching is performed, we speak of a *progressive* or *destructive search*.

In a *progressive search* we compute the middle point between the global lower and upper bounds of the makespan variable $C_{\max}$, i.e.,

$$M := \ell\mathrm{b}^{global}(C_{\max}) + \lfloor(\mathrm{ub}^{global}(C_{\max}) - \ell\mathrm{b}^{global}(C_{\max}))/2\rfloor.$$

Depending on the current local lower bound, we branch on the makespan variable if $\ell\mathrm{b}(C_{\max}) \neq M$. In the first branch we set $\mathrm{ub}(C_{\max}) \leq M$ and in the second branch we

set $\ell b(C_{\max}) \geq M+1$. This branching scheme has been used in [21] and is also called *dichotomizing search*.

In a *destructive search* we branch on the makespan variable in a way that we always try to find a schedule with the minimum possible makespan, i.e., $ub(C_{\max}) \leq \ell b^{global}(C_{\max})$ must hold. Using depth-first-search, either a feasible schedule is found or this makespan is rejected and we can increase the global lower bound on the makespan by one.

In case of RCPSP it turned out that a destructive search yields the best dual bounds if the presolving techniques from the previous section are not used. Whereas using the presolving techniques, there is only a slight difference in running time and number of nodes that are highly instance dependent, which means it cannot be decided from the computational results which technique is better to use. Hence, we do not discuss these results in detail on RCPSP instances. In Section 4.2, where we apply our techniques to labor-constrained scheduling problems, we will also compare the progressive and destructive search as well as a combination of both. There, half of the running time is spent using a progressive search, combined with a destructive search afterwards.

3.3.1.2 Generic branching schemes

Generic branching strategies analyze the impact of branching on the variables. Combinations of these and a short survey can be found by Achterberg and Berthold [3].

In *pseudocost-branching* a score per variable is computed that depends on the so far experienced objective improvement after branching in the corresponding direction. In the root node, when no branching has taken place so far, these values are uninitialized and cannot be used, or they are heuristically estimated.

In *strong branching* the variables are checked concerning which gives the best improvement in the objective function. Therefore, both branching decisions per variable are temporarily evaluated by imposing the corresponding bound and solving the LP relaxations. If all variables with fractional value are checked, we speak of *full strong branching*. Since this is too costly in terms of running time, only a subset with maximum cardinality is checked. Then, this rule is called *strong branching*. Developed by Applegate et al. [9] for solving TSP.

Pseudocost branching with strong branching initialization combines both ideas by performing strong branching on variables with uninitialized pseudo costs and then choose a variable according to the pseudocost score. *Reliability branching* has been introduced since over time the estimates on the pseudocosts from strong branching may have become weak. Hence, these values are reinitialized in this rule.

The *inference value* measures how many domain reductions could be performed after branching on a variable. There is a value for the up-branch and down-branch. Again, ideas from strong branching and reliability branching can be incorporated into this rule. In CP, yet another score depending on the frequency of propagation has been successfully applied. Michel and van Hentenryck [184] show that the *activity* score is competitive with several other CP-based branching schemes. The *activity* of a variable is increased in a node if a bound adjustment on this variable occurs. Before branching all scores of unfixed variables are subject to aging, i.e., they are scaled down. Then, the branching candidate is selected as a variable which has highest activity score.

In SAT the VSIDS-branching strategy is used, i.e., *variable state independent decaying sum*, see Moskewicz et al. [191]. A score (the vsids) of a variable is increased whenever

this variable is part of an explanation. This strategy prefers variables by which recent conflict clauses have been created. Another indicator is given by the *conflict length* of a variable, i.e., the average length of all clauses a variable appears in. The *cutoff value* measures the number of pruned subproblems after branching on a variable in the corresponding direction.

Achterberg and Berthold [3] combined the values for pseudocost, conflict values, conflict length, inference and cutoff values in the so-called *hybrid branching scheme*. The values are given per branching direction for each variable and are combined in a weighted sum. Then, the values for the up-branch and down-branch are combined by multiplication.

Node selection strategy

Another way to influence the search besides choosing the branching variable is given by the way the tree is traversed, i.e., which node is selected next. In CP and SAT, a depth-first search (DFS) is used often, while in IP the node with worst (estimated) dual bound is selected. Either this dual bound is computed from an LP relaxation or it is estimated by the gain induced on the branching variable. When applying DFS, the memory requirements for the tree are low (the tree is always a path). In IP, when switching from one sub-tree to another, several cutting planes and the LP relaxation itself must be re-constructed internally along the path from one node to the next. This often incurs additional switching times and huge memory requirements if only few nodes are cut-off this way. A good mixture between these two strategies is to perform a DFS with restarts, where restart does not mean that the whole branching tree is reset but that after e.g., 100 leaves of using DFS, a node with worst dual bound is selected and again a DFS is used for the next 100 leaves.

We use this technique in our study as it turns out to work well for RCPSP instances.

3.3.2 Computational study

We start this study by comparing the problem specific and generic branching schemes with each other. From these results we see that the generic branching schemes perform best on instances from PSPLib. We conclude our study by evaluating the impact of CP, IP and SAT techniques in our framework and compare these results to the best results reported in the literature.

3.3.2.1 Comparison of branching schemes

Problem-specific branching schemes have been used throughout the literature for RCPSP. One goal when using a CIP framework is to use generic branching schemes that learn throughout search from former branching decisions in order to guide the search into promising directions. Our experimental results show that for the RCPSP instances from PSPLib, generic branching schemes are a fruitful research direction whereas on the highly cumulative Pack instances, a problem-specific branching scheme outperforms other approaches. We turn on all presolving techniques and consider the following branching schemes: First, we use reliable pseudocost branching using the continuous relaxation from Section 2.4, denoted by 'relps'. Second, a CP-SAT hybrid search ('infer') is performed with inference branching, i.e., a score function is used that prefers vsids over

Table 3.6: Comparison of different branching schemes.

setting	nopt	bprimal	bdual	gap	avtime	avnodes
setS (114 instances)					(allopt: 56 instances)	
relps	110	**111**	112	0.19	2.48	2780.88
infer	110	**111**	**113**	0.15	**1.97**	**2471.75**
cum	80	90	81	5.93	7.22	13971.50
disj	95	107	97	3.32	18.45	44207.25
core	88	99	88	3.98	3.39	6249.68
lsdr	58	64	59	12.11	47.08	122769.41
setL (119 instance)					(allopt: 40 instances)	
relps	115	116	**119**	0.10	1.20	**335.20**
infer	**116**	**117**	**119**	0.10	1.15	458.68
cum	62	71	77	6.08	1.93	1924.42
disj	92	99	94	3.55	1.69	1918.50
core	83	94	91	2.74	1.18	512.55
lsdr	40	42	58	10.71	12.98	37049.55
Pack (55 instance)					(allopt: 32 instances)	
relps	39	44	52	1.43	1.54	1485.00
infer	41	46	51	1.48	1.50	2693.69
cum	**46**	**52**	**55**	**0.82**	1.05	**363.63**
disj	34	40	51	1.61	7.13	33355.59
core	40	49	51	1.37	1.91	3886.00
lsdr	39	46	53	1.20	2.07	7099.91

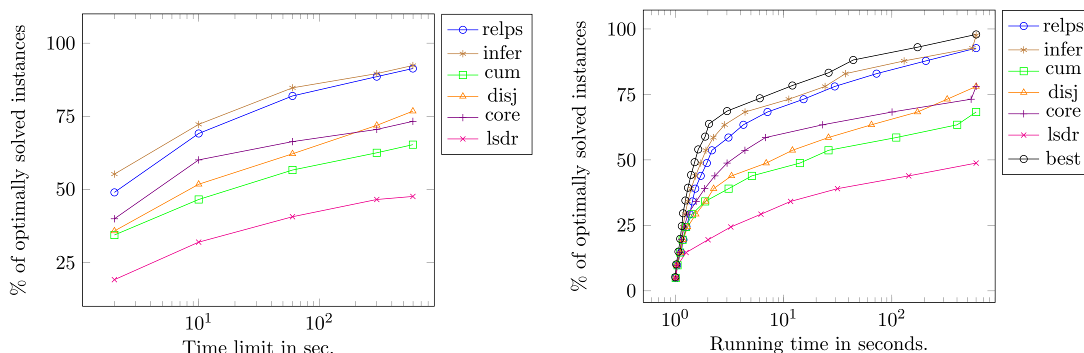


Figure 3.7: Number of optimally solved instances within given time limits of $2, 10, 60, 300$ and 600 sec. depending on the chosen branching rule.

inference values to guide the branching decisions. Third, in setting 'cum' branching candidates are selected that belong to an earliest peak of an earliest start resource profile. The branching variable has among all candidates the lowest ratio $\sum_k r_{jk} \cdot p_j / R_k$. Fourth, we branch by posting precedence constraints in setting 'disj'. In this case, we build an earliest start schedule and branch on the first pair of disjunctive jobs that are overlapping. Fifth, setting 'core' corresponds to branching on the first variable that needs to be shifted over its latest start time when building a list schedule with respect to the local lower bounds obeying the capacity. Last, branching according to the left-shift dominance rule is performed. This setting is denoted by 'lsdr'.

Table 3.6 in conjunction with Figures 3.7 and 3.8 shows that the generic branching rules outperform the problem-specific branching schemes on instances from PSPLib. At least 20 instances can be solved to optimality that could not be solved before while the

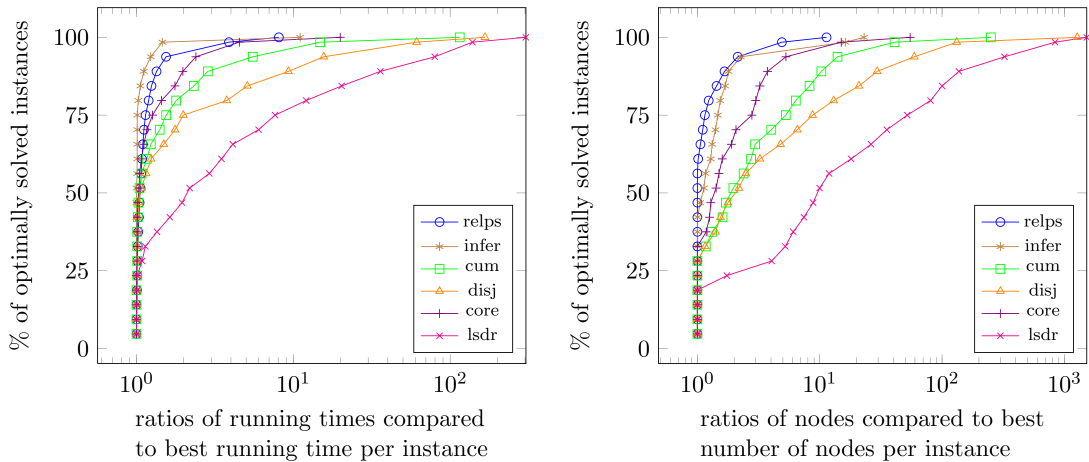


Figure 3.8: Ratios of nodes and running times given for all optimally solved instances by all branching rules.

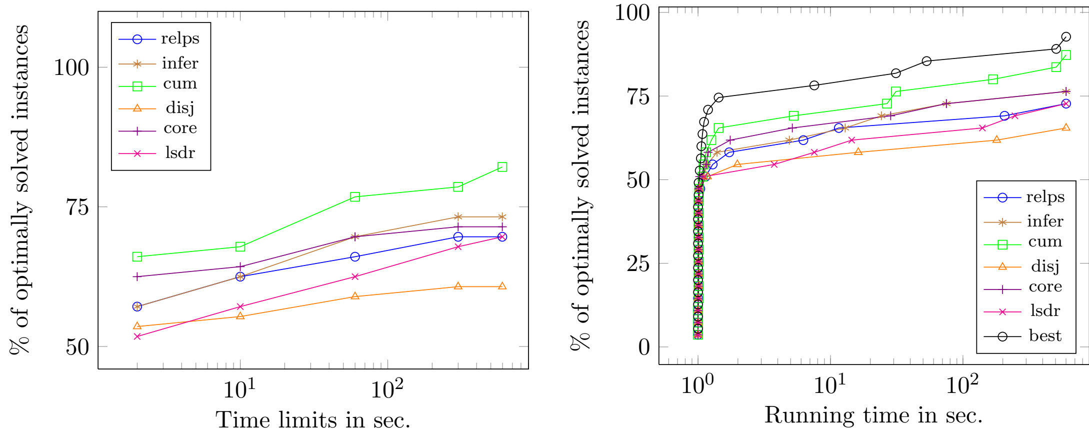


Figure 3.9: Impact of branching schemes on set Pack.

difference between the settings 'relps' and 'infer' are rather small. In particular, the final gaps with 0.1% are lowest for the generic branching schemes.

Branching by posting precedence constraints ('disj') yields on several instances better primal solutions than the other problem specific branching schemes but is not competitive at all with the generic branching schemes. In further experiments we restricted the set of instances to all those where at least 300 `disjunctive` constraints have been detected. Again, this branching rule was no better than the generic branching schemes. It turns out to be the best problem specific branching rule on PSPLib instances as most instances can be solved to optimality. Nevertheless, considering all optimally solved instances, the average running time (18.45 sec.) is much higher in this setting than by settings 'cum' or 'core'. This is mainly due to the larger number of nodes that need to be explored. Interestingly on setL, this branching rule performs much better than on setS because there are more disjunctive constraints present per instance.

Figure 3.9 and Table 3.6 clearly show that on highly cumulative Pack instances, the

cumulative branching scheme 'cum' outperforms even the generic branching schemes. It solves five more instances to optimality, yields the best dual bound on each instance and the lowest average final gap. Branching according to a core-heuristic ('core') also proves to be competitive with the generic branching schemes at least on those instances that can be solved within less than 100 seconds. Intuitively, this rule decides by branching which job to perform late. It adds a core in the first child node that hopefully yields more propagation as well as it creates free space during the execution of heuristics. This seems to be a good technique for cumulative but not for disjunctive instances.

Branching according to the left-shift dominance rule as done in setting 'lsdr' performs worst on PSPLib instances. The branching trees become huge and there is not much potential for propagation induced by this rule. Due to fewer jobs in the Pack instances, this rule does not perform that tremendously bad there. But it still needs three times as many nodes as the reliable pseudo-cost branching rule and about 20 times as many nodes as in setting 'cum'. On PSPLib instances, mainly the worse dual bounds are responsible for the bad results obtained by this rule, whereas on Pack instances the primal bounds are worse.

3.3.2.2 Comparison to other approaches

We compare our results with those reported by other authors. We impose a time limit of 600 sec. for this matches the other environments best. Other solvers from the literature and ours are given as follows.

- **LM** Pure CP approach by Liess and Michelon [177] where constraints are added similar to cuts in IP.
- **MCS** Lower bound is computed via minimal critical sets by Laborie [168].
- **DH** Branching scheme by Demeulemeester and Herroelen [84] using minimal delaying alternative, cut-set dominance rule and bound by Mingozzi et al. [186].
- **HOR** A pure SAT approach by Horbach [144].
- **SFS** Latest results obtained by Schutt et al. [227] using time-tabling edge-finding.
- **CP** Our procedure using time-tabling, overload-checking as propagation algorithms, a pure CP approach.
- **CPSAT** Our procedure using time-tabling, edge-finding and conflict analysis, a CP-SAT hybrid.
- **CIPc** Our CP-SAT procedure using the continuous relaxation.
- **CIPb** Our CP-SAT procedure using the relaxation by Pritsker et al. [209].

We do not show results by Dorndorf et al. [93], Klein and Scholl [159] or others as these are clearly outperformed by the latest approaches. Values for the results by MCS and DH are given as reported in [12] and for LM as reported in [177, 144].

Computational environments The environments of these solvers are as follows. Liess and Michelon (LM) use a NEC PowerMate running at 2GHz under Debian GNU/Linux with a time limit of 300sec; for the sets J90 and J120 we display the values with a time limit of 1,800 sec. Results for MCS are taken from [12] where a time limit of 1,800 sec was used on 1.4GHz, which corresponds to about 600 sec. on 2.0 GHz. The method has been implemented in ILOG Scheduler 6.1. The results for DH have been

obtained on IBM PS/2 Model P75 with a 80486 processor running at 25 MHz under Window NT. Though, this solver is pretty slow compared to others, it has been the best for a long time on set J30. Horbach (HOR) obtained his results on Dell Precision with Intel Core2Duo T6400 2.2GHz with 1GB of RAM running under Windows XP, compiled with Microsoft Visual Studio C++ 2005 compiler with a time limit of 300 seconds where no initial upper bound on the makespan is given. The machine is about twice as fast as the one used in LM.

For SFS we used the latest and best results obtained by Schutt et al. [227] who used different kinds of branching rules and problem formulations. Usually, the results for a time limit of 600 sec. are given for a 2.0 GHz run. The authors use an X86-64 architecture Intel Xeon E54052 processor with 2 GHz running under GNU/Linux compiled with the Mercury Compiler and grade hlc.gc.trseg.

Comparison As can be seen in Table 3.7, the methods DH and SFS are the only ones to solve all instances from the set J30 within the time limit. Results on the Pack instances have only been reported for MCS and SFS. Method DH has only been used on sets J30 and J60, due to the huge number of cut-sets. At the moment, SFS solves most of the instances from PSPLib among all settings. On the Pack instances, our CP-IP-SAT hybrid (CIPb) using the binary LP relaxation of Pritsker et al. [209] (see Section 1.3.1) solves more instances from set Pack which are the highly cumulative instances. Again, the results also depend on the employed branching scheme. Recall from Table 3.6 that a CPSAT-hybrid is able to solve 46 instances (83,6%) if branching decisions are performed according to the first resource conflict in an earliest start schedule. Combining this branching scheme with CIPb solved one instance less than CIPb with generic branching rule. Here the generic values are reported[2].

	LM	MCS	DH	HOR	SFS	CP	CPSAT	CIPc	CIPb
J30	97.7	97.9	**100.0**	97.3	**100.0**	97.2	97.9	98.5	96.9
J60	81.2	84.6	81.7	84.0	**89.6**	82.9	85.2	85.2	81.9
J90	78.8	79.4	–	79.0	**82.7**	78.3	80.0	79.8	78.5
J120	40.0	41.7	–	41.2	**47.0**	40.0	42.0	42.3	38.8
Pack	–	52.7	–	–	69.1	63.6	69.1	76.4	**78.2**

Table 3.7: Comparison of our method with those obtained in the literature. The percentage of optimally solved instances per test set is given.

Observe that all results from the literature have been obtained on different architectures. Our comparison of the CP, IP and SAT techniques is cleaner in the way that the same architecture and even the same branch-and-bound framework is used. The last four columns of Table 3.7 show that most instances from PSPLib are solved in setting CIPc, using the continuous relaxation in a CP-SAT hybrid. The SAT techniques play a crucial role as 2% additional instances (9 instances per set from PSPLib) can be solved in contrast to a pure CP approach. We compare our approaches more closely next.

[2]If branching on the first resource conflict is used, CPSAT solves 83,6% of the Pack instances.

3.3.2.3 Impact of CIP techniques

We now elaborate on the strength of the developed solving techniques in the unified framework. We use the results from the previous comparison to other approaches. Table 3.8 shows the numerical values for all four settings CP, CPSAT, CIPc and CIPb.

Table 3.8: Comparison of CP, CPSAT, CIPc and CIPb on all 2,095 instances.

setting	nopt	bprimal	bdual	gapRoot	gap	avtime	avnodes
J30 (480 instances)						(allopt: 464)	
CP	467	468	468	10.77	0.28	3.42	20800.26
CPSAT	470	471	473	10.77	0.18	1.41	1009.01
CIPc	**472**	473	**480**	10.70	**0.14**	**1.3**	**653.68**
CIPb	465	**480**	465	**10.06**	0.53	5.06	771.28
J60 (480 instances)						(allopt: 391)	
CP	398	411	435	8.46	3.24	4.02	20050.71
CPSAT	**409**	422	470	8.46	2.89	**1.36**	648.09
CIPc	**409**	426	**473**	8.41	**2.84**	**1.34**	**478.07**
CIPb	393	**466**	419	**8.23**	3.65	8.38	753.8
J90 (480 instances)						(allopt: 374)	
CP	376	425	449	6.81	4.02	2.85	9271.13
CPSAT	**384**	433	**475**	6.81	**3.80**	**1.12**	172.97
CIPc	383	440	472	6.79	**3.80**	**1.11**	108.64
CIPb	378	**466**	435	**6.72**	4.47	2.26	**107.62**
J120 (600 instances)						(allopt: 220)	
CP	240	426	531	16.07	10.94	7.24	32060.04
CPSAT	252	437	**583**	16.07	10.66	3.02	2234.06
CIPc	**254**	480	570	16.04	**10.43**	**1.47**	**574.28**
CIPb	233	**546**	462	**15.83**	11.32	25.72	875.96
Pack (55 instances)						(allopt: 33)	
CP	35	38	50	5.67	1.94	1.67	13904.27
CPSAT	38	41	50	5.67	1.77	**1.43**	2386.88
CIPc	42	45	**54**	5.67	1.37	2.0	776.27
CIPb	**43**	**54**	50	5.67	**1.22**	4.56	**368.24**

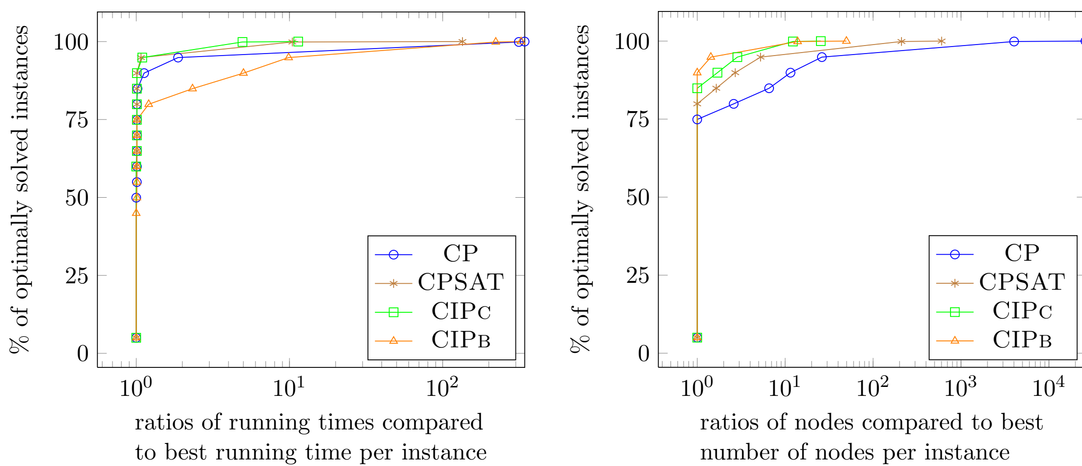


Figure 3.10: Distribution of running times and nodes for all solvers on all 2,095 instances.

From Table 3.8 we observe that the SAT part of the solver yields the highest improvements compared to its counterpart, the CP solver. The number of solved instances improves as well as the average number of nodes and running times on those instances that have been solved to optimality by all solvers. Here, the average number of nodes decreases by more than 95%, but the speed-up is only a factor between two and three.

Using the continuous relaxation from Section 2.4 on top of CPSAT in setting CIPc, we observe on sets J30, J60 and J90 that the average number of nodes further decreases by about 30% while the average running time remains the same. On instances from J120 and Pack, the number of nodes decreases to 1/3 on average. While on instances from J120 a speed-up factor of two is observed, the running time on the Pack instances increases by 1/3. This is due to the numerous cuts that are separated on Pack instances. Considering the final gap, we observe that CIPc obtains the best results except for the Pack instances, but the difference is rather small.

If the binary relaxation, represented by CIPb, is used, in total fewer instances can be solved to optimality from PSPLib than by using the CP approach. In contrast, CIPb solves most of the Pack instances. Similar results hold for the final gap. Here, CIPb gets worst results on instances from PSPLib and best results on instances from Pack. This solver needs the fewest nodes on average compared to the other solvers on Pack instances. CIPc is better on PSPLib instances. The distribution functions in Figure 3.10 represent the ratios of running time and nodes per instance compared to the best value obtained. We see that the running time in CIPb is often much worse, whereas when considering the number of nodes this approach yields the best results here. Interestingly, after applying all the proposed presolving techniques, there is almost no improvement in the dual gap if the binary relaxation is used.

3.3.2.4 A remark on improved vsids and reversed instances

Some of the computational results presented in this thesis have been rather confusing. We give few remarks on this.

In our studies on explanation algorithms from Section 2.3.3 on few instances we observed when using more sophisticated approaches to explain bound changes or infeasibilities, there were few instances where solving took much longer. After discussing this issue with the authors from [229], it turned out that their solver uses vsids per variable value, not per variable. Here, a score for the ub-branch and down-branch per value of an integer variable is used to guide the branching decisions. In their solver this way of collecting vsids is implicitly given by their underlying SAT model. This is one of the main differences between our solver and the one used in their experiments. In our experiments, where scores are collected per variable, we have made branching decisions based on possibly misleading values.

Initial experiments have been carried out in a new SCIP version[3]. In the latest release, scores per variable value can be used. This is bad in terms of running time if the integer variables have a high range but good on RCPSP instances where there are not too many values. This way, we are able to optimally solve between two and three more instances per set from PSPLib compared to the results shown throughout this thesis. In particular, the number of nodes per instance seems further decreasing, but the presolving and other

[3]SCIP 3.0 now contains a *Scheduler* example with our implementation of the `cumulative` constraint and a serial SGS as start heuristic.

techniques have also changed. We did not reimplement all experiments in the new solver. The new results are close to the other works but on few instances still worse.

Another experiment has been carried out on which the instances from PSPLib have been reversed. About the same number of instances can be solved from each set. Sometimes two or three instances can be solved more or less than before. Here, we observe that the number of nodes per optimally solved instance differs widely. On few instances more than 1,000 times as many nodes are needed than in the reversed instance, and vice versa. This observation also holds in the latest release, hence, the vsids seem not to be responsible for this. The first branching decisions made in a CP-SAT hybrid can be seen as a lottery ticket because the score values are uninitialized. Then, other score values are increased based on these first branches. This serves as a simple explanation. Another way of looking at RCPSP problems is to watch out for the *hard* parts of an instance. An instance may be more cumulative in the first part of the precedence network and rather disjunctive or even trivial in the last part. If the easy part is at the front, this part may be trivially fixed such that a much smaller instance remains. We cannot easily fix the last part of the precedence network as the makespan variable corresponds to the last job. Hence, looking at an instance from this perspective may be helpful in future work.

3.3.3 Concluding discussion

Solving instances of RCPSP remains a challenging task, though a huge amount of techniques has been developed yet. As discussed in Section 1.1.2 the hardness of such instances stems from different characteristics that are included in one model. Hardness of Bin Packing for highly cumulative instances, comes together with the inapproximable Graph Coloring characteristic that is hidden in highly disjunctive instances. Processing times larger than one and precedence constraints further complicate these characteristics.

The huge initial gaps of up to 60% on average on the unsolved instances that reach about 20% on average after 10 minutes of solving show that there are still many more efforts to be done in order to close these instances. The reason for these instances to be hard for CP-SAT hybrid solvers is that after the first branching decisions are made, not much propagation can be done. E.g., consider the case where a primal, say 100 units of time, and dual bound, say 80 units of time, with an initial critical path bound, of say 60, are given. Then, most propagation algorithms of the cumulative constraint are not able to adjust variable bounds if the processing times in these instances are smaller than ten. Then, the core-profile is empty, and no volume arguments may apply. Hence, many branchings must be performed—whatever branching scheme is used—until a conflict is detected. Then, the conflicts may also not carry much useful information for later propagation as other subtrees are totally different. This seems to be the main reason why on those instances none of the SAT and CP-SAT hybrid approaches (our approach, the one of Horbach [144] and the one of Schutt et al. [229]) is able to close the PSPLib instances containing 60 jobs. Even worse, a few instances with about 25 jobs from the set Pack which are highly cumulative are still not solvable.

We have seen in our study that using techniques from SAT, about 2% more instances can be solved from PSPLib compared to a pure CP approach within a time limit of 10 minutes. These are in particular hard instances. Furthermore, the number of solving nodes as well as the average running times highly decrease between 50% and 90%. Hence, the generic solving techniques that *learn* throughout search, by counting score values per variable, are a valuable step towards better understanding how to solve RCPSP

instances.

On the highly cumulative instances, our proposed preprocessing routines close several gaps that have not been closed before. These techniques should be a starting point for future work as they immediately reduce the search space by simple propagation of the precedence graph in the beginning as well as after branching. Redundant infeasible states are passed by this way.

On the unsolved, rather disjunctive instances from PSPLib, it can be observed that the resource profiles have a lot of free space to shift jobs in their time windows. This can certainly be used as a heuristic, that first (heuristically) adds precedence constraints between jobs and second uses an exact solver to solve the resulting instances faster. By using parallel computation this may lead to improved solution quality in contrast to genetic algorithms. For disjunctive instances even better exact results would be obtained if the conflict analysis tool-kit of a solver is able to use constraints within conflict clauses. Then, branching by posting precedence constraints may become stronger in CP-SAT hybrids. We performed experiments using this idea by adding the binary representation for pairs of disjunctive jobs. But the additional linear constraints and variables blew up the model that much that the solving times slowed down. In particular as several jobs can be executed in parallel, just posting precedence constraints does not suffice to reach a state where an earliest start schedule is resource feasible or infeasible. Further branching must be performed on resource conflicts induced by the non-disjunctive jobs.

A computational study on all different IP formulations known for RCPSP in a CIP framework is a valuable next step. As observed in several former studies, the cut-set dominance rule by Demeulemeester and Herroelen [84] only works well on small instances as the number of cut-sets grows too fast. Either controlling this number or extending the technique to subschedules, give another direction for future work. In general, schedule partition procedures that optimally solve subschedules (which is possible due to a smaller size) and then combining these subschedules to an optimal schedule of the whole instance might be useful even for large instances from practice. These ideas, from generalizing dominance rules to schedule partition procedures, go well with the fact that some RCPSP instances are much easier to solve if the schedule is simply reversed. Parts of a scheduling instance can be easy for themselves, but may lead to an unnecessary overhead induced by bad branching decisions or due to useless conflict clauses.

The intriguing hardness of RCPSP and the given benchmark instances that are still unsolved nowadays will attract more and more researchers over the next decades. While this thesis sheds some light on the importance of the CP, IP and SAT solving techniques, it hopefully inspires more researchers to deal with generic solvers and to develop even better presolving and search techniques.

Chapter 4

Applications of the CIP framework for RCPSP

In this chapter we apply the techniques developed so far for RCPSP to two related problems. First in Section 4.1, we consider the resource-constrained project scheduling problem with discounted cash flows where a non-linear objective function is introduced into the RCPSP model. We are particularly interested in the impact of the continuous relaxation that is proposed in Section 2.4 as now there is more than one variable with non-zero coefficient in the objective function. We will see that if as well positive as well as negative coefficients are part of the objective function, the CIP approach based on the continuous relaxation outperforms the classical CP approach. In particular, the dual bounds are much stronger on such instances compared to a pure CP approach.

Second in Section 4.2, we consider another RCPSP related problem in which the resource demands per job vary over time, called *labor-constrained scheduling problem.* Practical instances for these types of problems are taken from the literature and we evaluate the different branching schemes, as well as CP, CP-SAT and CIP approaches in this context. Interestingly, the SAT part becomes less important as the generated conflicts are less reusable and possibly the vsids are less effective in this context.

4.1 Application to RCPSPDC

In several scheduling applications, makespan minimization as in classical RCPSP is not the major goal. Either because a reasonable makespan is known by which all jobs shall be completed or because the main cost drivers are independent of the completion of the project. Often, it must be decided whether job is taken into the schedule or not, like in just-in-time-production. Then, the objective function models the profit for each job that is processed.

Another example can be found in an application from the mining industry, where high cost occur for setting up a stope until the valuable goods, such as ore-bodies, are reached. Scheduling decisions must be made in order to perform profitable tasks early (to generate monetary values). Hence, financial considerations must be taken into account when creating a schedule. For each job or event a cash flow is specified that is to be paid (cash out-flow) or is gained (cash in-flow) at the occurrence of an event or at the execution of a job (start or end time). Events may be fixed points in time, or depend on the progress of the project. Since the value of money decreases with time, the cash flows

are discounted. The goal is to maximize the *net present value*, i.e., the sum over cash in-flows minus cash out-flows. This problem is known as *Resource-Constrained Project Scheduling with Discounted Cash Flows*, or RCPSPDC for short.

We consider real-world problems as they arise in the mining industry on a strategical level of planning. Besides data from publicly available benchmark instances, we obtained real world data by Chris Alford[1] on which we evaluate our algorithm. In this application, positive cash flows model the profit of extracted ore bodies and negative cash flows model the payments to be done to set up the stope, for grabbing and several other logistic necessities. Often payments have to be done first, before cash in-flows can be obtained, i.e., first we need to set up and start grabbing until valuable material is reached.

We present the additional requirements for our CIP framework for RCPSP in order to solve resource constrained project scheduling problems with the objective to maximize the net present value. This adapted framework is able to compute optimal solutions if enough time and memory are at hand. Since this is not the current status in practice, a heuristic is incorporated that uses the values of LP or CP solutions as a basis for a serial SGS (see Section 1.4).

We first give a formal problem description and an overview on the research on RCPSPDC. Next, we present the CIP-based solution approach in which we iteratively approximate the non-linear objective function via linear equations. Propagation rules, separation rules, branching strategies and heuristics are then presented in order to speed-up the branch-and-bound search. In the last section, we evaluate our approach on standard benchmark instances and on the new data from practice.

4.1.1 Problem description

In Resource-Constrained Project Scheduling with Discounted Cash Flows (RCPSPDC), we are given a set $\mathcal{J}$ of non-preemptable jobs and a set $\mathcal{R}$ of renewable resources. Each resource $k \in \mathcal{R}$ has a bounded capacity $R_k \in \mathbb{N}$. Every job j has a processing time $p_j \in \mathbb{N}$ and resource demands $r_{jk} \in \mathbb{N}_0$ per resource $k \in \mathcal{R}$. The start time S_j of job j is constrained by its predecessors that are given by a precedence graph $G = (\mathcal{J}, A)$. An arc $(i, j) \in A$ represents a precedence relationship, i.e., job i must be finished before job j starts. We assume that a first and a last dummy job exist that model the start and end of the project. In a schedule, i.e., an assignment of integer start times S_j for each job j, at each point in time, the cumulative demand of the set of jobs running at that point, must not exceed the given capacities.

$$\sum_{j: S_j \leq t < S_j + p_j} r_{jk} \leq R_k \qquad \forall\, t, \forall\, k \in \mathcal{R}$$

The objective is to maximize the net present value (NPV), i.e., $\sum_j f(S_j)$ where the function $f : \mathbb{R} \to \mathbb{R}$ is determined by the cash flow $c_j \in \mathbb{R}$, a discount rate $\delta \in (0, 1)$ and is defined as:

$$f(S_j) := c_j \cdot e^{-\delta(S_j + p_j)}. \tag{4.1}$$

The goal in RCPSPDC is to schedule all jobs with respect to resource and precedence

[1]We thank Chris Alford, Director at Alford Mining Systems, Melbourne, Australia, for providing us the real-world data.

constraints, such that the net present value is maximized:

$$\begin{aligned}
\max \quad & \sum_{j \in \mathcal{J}} c_j \cdot e^{-\delta(S_j+p_j)} & \\
\text{subject to} \quad & S_i + p_i \leq S_j & \forall\, (i,j) \in A \\
& \sum_{j:S_j \leq t < S_j+p_j} r_{jk} \leq R_k & \forall\, k \in \mathcal{R}, \forall\, t \\
& S_j \in \mathbb{N}_0 & \forall\, j \in \mathcal{J}.
\end{aligned}$$

Observe that initially no upper bound on the makespan is imposed on these instances. If only negative cash-flows occur at the end of the project, such jobs will not be executed in finite time in an optimal solution. Closing the stope is a typical example of such a job that must be performed at the end. Hence, in our study we impose an upper bound on the makespan.

4.1.2 Related work

RCPSPDC has been well studied and heuristic as well as exact approaches exist. The problem is classified as $PS|prec|\sum C_j^F \beta^{C_j}$ [36] or $m, 1|cpm, \delta_n, c_j|npv$ according to the notation scheme of Herroelen et al. [140]. If no resource constraints are taken into account, the problem is called *Payment Scheduling Problem* [129] if cash flows occur at specified events (that may not correspond to jobs) and *Net Present Value Problem* [185] if cash flows are linked to the start or completion times of the jobs. Payment models as they occur frequently in practice are presented by Ulusoy et al. [252], which cover lump-sum payment, i.e., payments only occur at the end of the project, payment at event occurrences, payment at equal time intervals with a final payment at the end and progress payment where payments are done in regular time intervals until the project is completed.

Smith-Daniels and Aquilano [241] compare priority based list scheduling rules. They show that latest start policies lead to better NPV and lower project duration than earliest start schedules for instances from the Patterson test set containing 110 instances. Ulusoy and Özdamar [200] present forward-backward scheduling based heuristics, whereas Baroum and Peterson [25] evaluated single- and multi-pass procedures that are used by project planners. There, positional weight heuristics outperform former priority based rules. Icmeli and Erengüc [147] perform a tabu search on earliest start schedules by shifting jobs by one time unit and evaluating the outcome. Zhu and Padman [271] run different priority based heuristics in parallel (distributed computing). A combined heuristic (*asynchronous team*) learns from the outcomes of the single runs to get better priority based lists.

Mika et al. [185] present numerous heuristic approaches using genetic algorithms, ant colony optimization and tabu search. Selle and Zimmermann [234] propose a bi-directional heuristic that combines ideas from forward- and backward-scheduling. Vanhoucke [257] presents a scatter search procedure (see [121, 181]) i.e., an evolutionary algorithm, that combines solutions from an initial population to new solutions and performs improving steps on them. Ulusoy et al. [252] develop a genetic algorithm for the multi-mode RCPSPDC with renewable, non-renewable and doubly-constrained resources

that is able to deal with different types of non-linear cost functions. Vanhoucke [256] presents a genetic algorithm for RCPSPDC in which the makespan is not fixed, but can be violated by incurring penalty costs. He compares different mutation and crossover operators on a huge set of benchmark instances.

Goto et al. [126] present a multi-pass meta-heuristic (a two-stage tabu search) for a generalized multi-mode RCPSPDC. By a mode they model how a job is completed (e.g., different resources) and the amount of allocated resources then determines the processing time. Furthermore, they use additional payment models, e.g., they consider time-dependent equipment cost. In the first stage of their heuristic, a feasible solution is created and in the second stage it is improved by shifting the jobs. A key idea in their algorithm is to allow temporarily infeasible schedules. Vanhoucke and Debels [258] investigate NPV problems with time-switch and other side constraints. In 2010, an ant colony optimization algorithm for multi-mode resource constrained project scheduling with discounted cash flows has been presented by Chen et al. [57] which performs good on instances with up to 100 jobs, even better than genetic algorithms, simulated annealing or tabu search.

Besides heuristic solutions, solution quality guarantees are desired from the management perspective. One way to obtain dual bounds is to neglect the resource constraints, which results in the NPV problem. Dual bounds have been derived for that problem since the 1970s, via LP methods, see [129, 214], or via so-called *early trees* [105]. Demeulemeester et al. [86] propose an optimal recursive procedure. In each iteration, a job with positive (negative) cash flow is left- (right-) shifted in order to improve the objective value. When shifting more than one job, this gain must be re-calculated in each iteration in order to obtain a complete method which leads to higher solving times. Schwindt and Zimmermann [231] generalize this method to networks in which generalized precedence constraints are present. Starting with an earliest start schedule, they perform improving steps. In such a step, they iteratively compute a steepest ascending direction and use a line search to obtain a new feasible solution in that direction.

Doersch and Patterson [91] present one of the first IP formulations wherein Smith-Daniels [242] incorporates material management cost. They are able to solve instances with 15 to 25 jobs. Patterson et al. [204] present an algorithm based on backtracking that works in combination with a right-shifting heuristic. The algorithm has been evaluated on a test set consisting of 91 instances with 10 to 500 jobs per instance, but the algorithm was only able to solve small instances. Another IP approach for RCPSPDC goes back to Yang et al. [267], who show that instances with up to 30 jobs can be solved to optimality. In their approach, they apply a depth-first search strategy, present a heuristic and prune the search tree by identifying dominated nodes based on network cuts.

Vanhoucke et al. [259] present a branch-and-bound algorithm that uses the early trees to derive dual bounds and they use a branching scheme that relies on minimal delaying alternatives in order to resolve resource conflicts. They resolve these conflicts by branching into children and thereby posting new precedence constraints, a technique that has been earlier employed by Icmeli and Erengüc [147].

Kimms [158] shows that dual bounds for RCPSPDC can be efficiently computed via Lagrangean Relaxation. He uses the solution values of the relaxation as a basis for a parallel scheduling scheme and obtains good lower and upper bounds for instances with up to 120 jobs.

Vanhoucke [254] provides a huge amount of benchmark instances tested on various

settings which we will use in our computational study. In [256, 257] these instances have been used to evaluate the heuristics. The first test set contains 1,800 instances, 360 instances with 10, 20, 30, 40 and 50 jobs, respectively. Most instances with up to thirty jobs are well solved with the method by Vanhoucke et al. [259]. The second set contains instances with 25, 50, 75 and 100 jobs. But as our results and those of [226] reveal, the net present values as reported in [254] are often wrong.

Liu and Wang [178] present a CP based approach for a construction project in which the resource levels (capacities) are part of the objective function. With their lazy clause generator, Schutt et al. [226] can close almost all of the small instances from Vanhoucke [257], but they do not report any results on the larger sets with 40 and 50 jobs. Schutt et al. [226] study propagators for the objective function, they use the linearized LP formulation by [129] and an improved version of the early trees by [86]. With this approach they are able to solve almost all instances from [254] which contain up to 30 jobs and different percentages of positive and negative cash flows. Only on about 1% of the instances no optimal solution is found or proven. Unfortunately, the authors did not run the experiments on the larger set with up to 100 jobs.

4.1.3 Solution approach

In this section, a constraint integer programming (CIP) model that is based on the generic implementation of the cumulative constraint from Section 2.1.3 is presented. Based on this model, we adapt a list scheduling heuristic in order to obtain good primal solutions.

4.1.3.1 Model

In RCPSPDC we need to find a precedence- and resource-feasible assignment of start times S_j that maximizes NPV. We model the resource-constraints via the `cumulative` constraint, and the precedence constraints via `precedence` constraints, see Section 2.1. To model the non-linear objective function, we introduce for each start time variable S_j a continuous variable X_j and an `npv`-constraint. This constraint gets as input the start time variable S_j, a cash flow value c_j, a discount rate $\delta \in (0,1)$, a translation t and the objective variable X_j. Intuitively, the translation t corresponds to the point in time, when the payment must be done after the start time of the job. We will use $t := p_j$ in our study since there cash flows arise at the completion of the jobs. Hence, we can define the npv-constraint as follows:

$$\texttt{npv}(S_j, X_j, c_j, \delta, p_j) \quad := \quad \left\{(S_j, X_j) \in \mathbb{N}_0 \times \mathbb{Q} \mid X_j = c_j \cdot e^{-\delta(S_j+p_j)}\right\}. \tag{4.2}$$

Given this, a CIP model reads as:

$$\begin{array}{lll} \max & \sum_{j\in\mathcal{J}} X_j & \\ \text{subject to} & \texttt{precedence}(S_i, S_j, p_i) & \forall\,(i,j) \in A \\ & \texttt{cumulative}(\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}_{.k}, R_k) & \forall\, k \in \mathcal{R} \\ & \texttt{npv}(S_j, X_j, c_j, \delta, p_j) & \forall\, j \in \mathcal{J} : c_j \neq 0 \\ & D(S_j) = \mathbb{N}_0 & \forall\, j \in \mathcal{J} \end{array}$$

4.1.3.2 Propagation

We use domain propagation to synchronize the start time variables with the continuous variables that appear in the objective function. Whenever the bounds of a variable S_j change, this decision is propagated to X_j according the following lemma.

Lemma 4.1. *Given a variable S_j with bounds $\ell\text{b}(S_j) \leq S_j \leq \text{ub}(S_j)$, a cost function f (4.1) with cash flow c_j, and a variable X_j for which $X_j = f(S_j)$ must hold. Then the following bound adjustments can be made:*

$$c_j > 0 \implies f(\text{ub}(S_j)) \leq \ell\text{b}(X_j) \leq X_j \leq \text{ub}(X_j) \leq f(\ell\text{b}(S_j))$$

and

$$c_j < 0 \implies f(\ell\text{b}(S_j)) \leq \ell\text{b}(X_j) \leq X_j \leq \text{ub}(X_j) \leq f(\text{ub}(S_j))$$

Similarly, changes on variables X_j can be used to change the domain of S_j. Due to floating point arithmetics, these updates need to be handled carefully.

Explanations for the propagation rule In a CIP framework, the analysis of infeasible search states plays an important role. Therefore, we also deliver explanations, which are straight-forward using Lemma 4.1. If the lower bound of X_j has been changed by the npv-constraint, the reason is the lower (upper) bound of S_j if the cash flow is negative (resp., positive). A similar rule holds for upper bound changes.

4.1.3.3 LP relaxation

LP relaxations provide dual bounds and enable the solver to use branching strategies that incorporate a polyhedral view. In contrast to pure makespan minimization problems, such as RCPSP, this view becomes more interesting in case of RCPSPDC since now the variables occur in the objective function. We use a continuous relaxation of the cumulative constraint that contains cuts on the start time variables:

$$\sum_{j \in J} S_j \geq W,$$

for a set $J \subset \mathcal{J}$ and a constant W that need to be computed, see Section 2.4.

Now, we describe the linear relaxation of the NPV-objective used. We consider the non-linear function $f(S_j) = c_j \cdot \exp^{-\delta(S_j + p_j)}$ for every job j. In case that $c_j > 0$, a secant through the points $(\ell\text{b}_j, f(\ell\text{b}_j))$ and $(\text{ub}_j, f(\text{ub}_j))$ gives the best upper bound that can be expressed by linear inequalities on that function.

$$f_1(S_j) = f(\ell\text{b}(S_j)) + (S_j - \ell\text{b}(S_j)) \cdot \frac{f(\text{ub}(S_j)) - f(\ell\text{b}(S_j))}{\text{ub}(S_j) - \ell\text{b}(S_j)} \tag{4.3}$$

Corollary 4.2. *If $c_j > 0$, then a valid inequality on the profit of job j is given by*

$$X_j - \frac{f(\text{ub}(S_j)) - f(\ell\text{b}(S_j))}{\text{ub}(S_j) - \ell\text{b}(S_j)} S_j \leq f(\ell\text{b}(S_j)) - \ell\text{b}(S_j) \cdot \frac{f(\text{ub}(S_j)) - f(\ell\text{b}(S_j))}{\text{ub}(S_j) - \ell\text{b}(S_j)}.$$

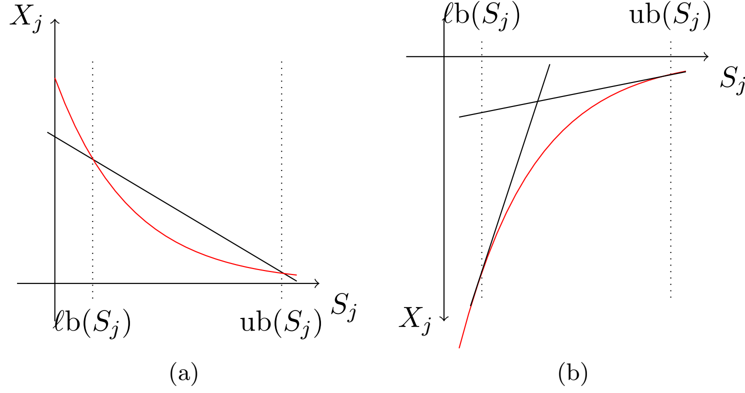


Figure 4.1: Linear approximation of npv-objective for local bounds.

If $c_j < 0$, the cost function can be described arbitrarily close by a set of tangents. We only introduce tangents in the points $(\ell b(S_j), f(\ell b(S_j)))$ and $(ub(S_j), f(ub(S_j)))$ by using the first derivate $f'(S_j) = -c_j \cdot \delta e^{-\delta(S_j+p_j)}$. Doing so ensures that any solution in which the start variable S_j takes the value of its local bounds, the correct global bound is set. The formulas are:

$$f_2(S_j) = f'(\ell b(S_j)) \cdot (S_j - \ell b(S_j)) + f(\ell b(S_j)) \tag{4.4}$$

$$f_3(S_j) = f'(ub(S_j)) \cdot (S_j - ub(S_j)) + f(ub(S_j)). \tag{4.5}$$

Corollary 4.3. *If $c_j < 0$ holds, then valid inequalities on the profit of job j are given by*

$$X_j \le f_2(S_j) \qquad \text{and} \qquad X_j \le f_3(S_j).$$

Figure 4.1 visualizes the linearization. A closer approximation of the cost curve can be obtained by adding several tangents in the interval $[\ell b(S_j), ub(S_j)]$. These tangents are left to the separation routines after branching and hence are automatically separated.

Our separation procedure works as follows. Given the local bounds of S_j and an LP solution, if X_j violates any equation of $X_j \le f_i(S_j)$ for $i = 1, 2, 3$, depending on the value of c_j, then the corresponding inequalities are separated.

4.1.3.4 Heuristic

We use a list scheduling algorithm as presented in Section 1.4 that first sorts the jobs, e.g., by their start times of an arbitrary solution, and then schedules them in that order with respect to resource and precedence constraints. This heuristic is called in various stages, during presolving, where the bounds of the variables are used as an order (lower bounds, upper bounds, and convex-combinations) and after the LP has been solved in any node of the branch-and-bound tree, in this case the jobs are sorted according to the start times in the LP-solution. This yields, the so-called *forward-schedule*. If a feasible forward-schedule is found (which is not always the case due to resource conflicts and tight project deadlines), a *backward-schedule* is computed. We denote by S'_j the start time of job j in that schedule. The jobs are scheduled in non-increasing order of completion times $S'_j + p_j$. Hence, when we consider a job, all of its successors have already been scheduled and its upper bound can be updated according to them. A job

in that order is scheduled as late as possible with respect to resource constraints in the interval $[S'_j, \text{ub}(S_j)]$ if it has negative cash flow. If a job has positive cash flow, it should be scheduled early – we schedule such a job as early as possible in the interval $[S'_j, \text{ub}(S_j)]$ with respect resource constraints.

It is clear from the way the algorithm works, that a feasible solution will be found in the second phase if one has been found in the first stage, since no job needs to be scheduled before its start time in the first schedule.

4.1.3.5 Branching

As seen in the preceding chapters, RCPSP can be solved via a destructive search on the makespan such that a pure feasibility problem needs to be solved until a feasible solution is found. Constraint Programming tools and SAT solvers perform outstandingly well on such problems. Introducing the NPV-objective into the model changes the nature of the problem completely. Not only a feasible solution for some given makespan needs to be found, but among all such feasible solution an optimal one.

We branch on variables where the current LP solution violates the real profit most. This helps to introduce stronger cuts for the objective function early and intuitively leads the search into promising spaces where to find good primal solutions. This is a counterpart to the common branching strategies in SCIP that store branching histories per variable (including pseudo-cost, inference and conflict scores).

4.1.4 Computational study

Benchmark instances For our benchmarks, we use instance sets from Vanhoucke [254] which contain each 180 instances with 25 (npv25), 50 (npv50), 75 (npv75) and 100 (npv100) jobs per instance. Each instance contains up 4 cumulative resources with capacity 10 and each job has a resource demand and processing time between 1 and 10. Integer cash flows that are randomly generated from $[-500, 500]$ for each job are given. The discount rate δ is set to 1% in all experiments.

We additionally generated artificial instances with $1,000$, $2,000$, $5,000$ and $10,000$ jobs per instance by concatenating the instances with 100 jobs and deleting the dummy start and sink nodes. We denote theses sets by npv1000, npv2000, npv5000 and npv10000. Cash Flows are i.i.d. chosen at random from $[-500, 500]$.

Initial experiments showed that using time-table edge-finding slowed down the solution process too much. Hence, we only apply time-tabling and overload checking in each node of the search tree to propagate the resource constraints. In our experiments with the instances from Vanhoucke [254] with up to 100 jobs per instance, our solution values differed from the reported ones. Schutt et al. [226] also run their experiments on the sets npv25-100 (but no results are explicitly given) and it turned out that the given primal and dual bounds are wrong. Hence, we cannot give a clean comparison to the best values from the literature on these instances. Other instances sets are provided by Vanhoucke [254] which contain much fewer jobs and are used by Schutt et al. instead. But these sets contain only up to 30 jobs and are therefore less suited while heading for large instances. Hence, we stick to the test set containing up to 100 jobs and created additional larger sets as described above.

Results for makespan minimization Table 4.1 shows the makespans for the large instances obtained within one hour. An initial solution was in most cases much worse and is used in our experiments as initial makespan, which is displayed in column "MS". We remark that after one hour for instance set npv10000, the gap is always larger than 100% and 15,000 nodes in the shifted geometric mean can be explored. On set npv5000 the gap is also always larger than 100% and about 37,000 nodes have been explored in the shifted geometric mean.

Table 4.1: Minimum makespans computed within one hour and the corresponding lower bound. "MS" denotes an initial makespan that will be used in the experiments (A simple list scheduling finds a feasible schedule).

instance	dual	$C_{\max}$	MS	instance	dual	$C_{\max}$	MS
10001	481	943	970	50001	2240	4716	4900
10002	634	2178	2250	50002	3961	11211	12000
10003	634	2178	2250	50003	1540	4031	4090
10004	882	1653	1680	50004	1751	3709	4000
10005	863	2255	2400	50005	2913	10859	11500
10006	332	800	820	50006	1751	3709	4000
10007	408	881	890	50007	2240	4716	4900
10008	408	881	890	50008	1622	3999	4200
10009	350	798	830	50009	3961	11211	12000
100010	394	771	800	500010	1972	4414	4500
20001	922	1891	1950	100001	4431	9419	9800
20002	1712	3305	3375	100002	7858	22471	24000
20003	643	1609	1635	100003	3040	8025	8500
20004	670	1595	1675	100004	3448	7688	7900
20005	921	1891	1950	100005	5752	21701	23000
20006	1197	4346	4600	100006	3448	7688	7900
20007	1712	3305	3380	100007	4429	9419	9750
20008	1615	4481	4750	100008	3219	7995	8300
20009	1197	4346	4575	100009	7858	22471	23800
200010	811	1762	1790	1000010	3921	8821	8950

Comparison of RCPSP and RCPSPDC The given instances from Vanhoucke [254] come with their own makespan. We use the given makespans for our computational results. Since it is easy to change the objective function back to makespan minimization, we also run our framework on these instances as RCPSP. Table 4.2 shows how many instances can be solved to optimality ("nopt"), for which a gap remains ("nfeas") and which cannot be solved ("nosol") within a time limit of ten minutes for RCPSP and RCPSPDC. In case of RCPSPDC, the settings with 0% negative cash-flows and a deadline extension of 5 for npv25 and npv50 and of 10 for npv75 and npv100 have been chosen for comparison. As expected, problems with a non-linear objective function are much harder to solve and sometimes even no feasible solution will be found, a detailed comparison is discussed later in Table 4.4.

Besides that RCPSP is a hard problem in terms of complexity theory and also from a computational point of view, the non-linear objective function of RCPSPDC makes

Table 4.2: RCPSP vs RCPSPDC computations for the small test sets within a time limit of 10 minutes.

	RCPSP			RCPSPDC		
instance set	nopt	nfeas	nosol	nopt	nfeas	nosol
npv25	98	82	0	90	90	0
npv50	33	147	0	14	133	33
npv75	11	197	0	0	110	70
npv100	8	172	0	0	63	117

the problem even more challenging. Table 4.3 shows the number of branching nodes per second that can be created during search on average over a set of instances. On average, when doubling the number of jobs, this results in less than half the number of nodes per second. On the other hand, the number of nodes that can be explored per second for RCPSP is five to 30 times as large as for RCPSPDC. Hence, for RCPSPDC the computational efforts are tremendously higher than for RCPSP.

Table 4.3: Number of branch-and-bound nodes that can be explored in one second on average for the different sets of instances. Sets npv25-100 have a time limit of ten minutes and sets npv1000-10000 have a time limit of one hour for RCPSP and four hours for RCPSPDC.

instance set	RCPSP nodes/sec	RCPSPDC nodes/sec	instance set	RCPSP nodes/sec	RCPSPDC nodes/sec
npv25	3993.2	846.4	npv1000	65.3	2.3
npv50	1877.6	323.1	npv2000	30.9	1.2
npv75	1089.3	151.7	npv5000	10.6	0.9
npv100	719.4	76.2	npv10000	4.2	0.6

Results for RCPSPDC: CP vs. CIP Next, we compare the results of a CP versus a CIP approach on RCPSPDC instances. It is commonly believed that pure feasibility problems are well solved via CP search (in combination with SAT), whereas a CP framework behaves much worse if the objective function has a bigger involvement.

In Table 4.4 we compare the number of optimally solved instances ("nopt"), the number of instances where a feasible solution has been found but is not proven to be optimal ("nfeas") and the number of instances where no feasible solution has been found ("nosol"). Column "bprimal" denotes how often the best primal bound and "bdual" how often the best dual bound have been found by the corresponding setting. These numbers are computed over all sets of instances with $0\%, 60\%$ and 100% negative cash-flows. Instances from sets npv25 and npv50 have a deadline extension of 5, whereas instances from sets npv75 and npv100 have a deadline extension of 10.

Figure 4.2 compares how often a CP or a CIP approach finds the best primal or dual bound depending on the percentage of negative cash flows. The results show that the CP search behaves good on homogeneous instances where all cash-flows are positive or negative. As soon as we mix these (60%), the frequency by which the best primal and dual bounds are obtained via the pure CP search noticeably decrease. In particular, a CIP

search wins by far in columns "bprimal" and "bdual" for the settings with 60% negative cash-flows. If all cash-flows are non-negative (0%) the CP search wins in these two columns. For non-positive cash-flows (100%) the CIP search wins in column "bprimal" and the CP search in column "bdual". Here, the search strategy (using pseudo-costs in CIP and branching on LP solutions) play an important role and lead to this behavior.

The results conform the hypothesis that CP techniques work well for feasibility problems and for easy objective functions. But as soon as these are more intriguing, the IP part of a solver becomes more important and helps drastically by guiding the search in better directions and pruning unpromising nodes.

Table 4.4: Comparison of solved instances and primal and dual bounds that have been obtained by a pure CP in contrast to a CIP approach on RCPSPDC instances.

		CP					CIP				
set	%neg.cf	nopt	nfeas	nosol	bprimal	bdual	nopt	nfeas	nosol	bprimal	bdual
npv25	0	90	90	0	150	146	95	85	0	132	124
npv25	60	83	97	0	94	102	93	87	0	166	157
npv25	100	79	101	0	104	164	74	106	0	148	86
npv50	0	14	133	33	106	143	15	123	42	62	51
npv50	60	2	150	28	37	4	8	159	13	133	177
npv50	100	11	156	13	77	148	11	155	14	108	42
npv75	0	0	110	70	75	147	3	103	74	44	33
npv75	60	0	122	58	32	0	0	123	57	103	180
npv75	100	0	138	42	58	123	3	146	31	99	57
npv100	0	0	63	117	40	151	0	61	119	23	29
npv100	60	0	64	116	8	3	0	64	116	58	177
npv100	100	0	70	110	27	110	0	77	103	57	70

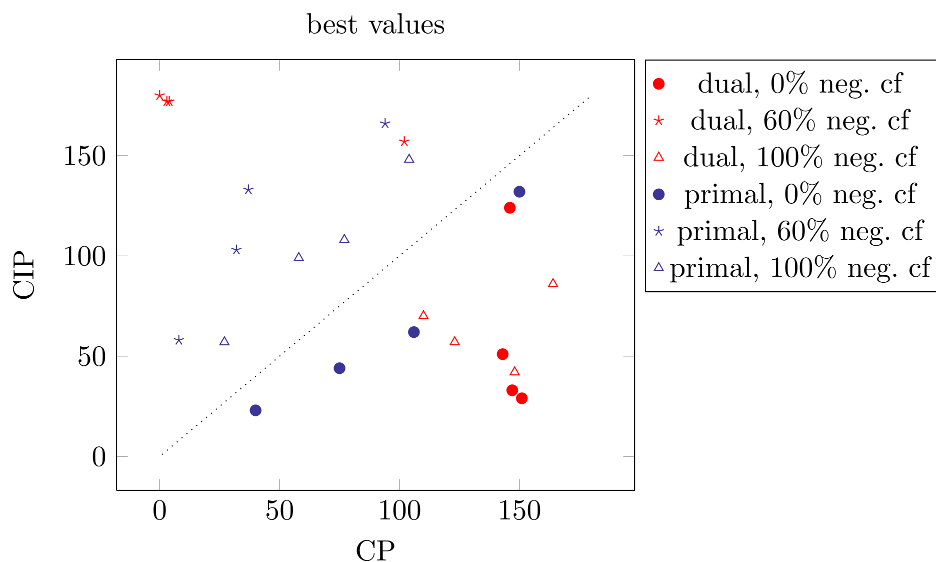


Figure 4.2: Comparison of the number of best dual and primal bounds for sets npv25, npv50, npv75 and npv100 depending on the percentage of negative cash flows per instance.

Table 4.5: Results for industrial instances. For each instance the chosen makespan and the NPV after one and five hours of computation time are given for the CP and CIP approach. The last column shows the best dual bound on NPV.

		CP		CIP		
instance	makespan	1h	5h	1h	5h	best dual
Z	8171	84989158	109850185	167064399	167064399	–
Ch	7251	1745626754	2096174368	2154911393	2179688393	3692427138
GL	2196	992091900	994128410	993453641	993453641	1097410242
LoA	2339	5517200536	5517200536	5517200536	5517200536	5560732357
NZ_def	3014	987036673	987036673	977032853	1036880107	1233895270
NZ	1918	1218365992	1218365992	1218365992	1218365992	1234406002
W	3417	475275599	475275599	466596268	466596268	609485334
ZF	6384	163457366	163457366	228664414	228664414	555171615

Results for industrial instances Eight industrial instances[2] that have been generated from real projects are given, denoted by Z, Ch, GL, LoA, NZ_def, NZ, W and ZF. During the generation process, processing times have been rounded on a daily basis and resource demands have been scaled to be integral. The instances contain between 2 (W) and 36 (NZ,LoA) resources and between 1410 (NZ) and 11769 (ZF) jobs. Approximate cash flow values are given. A 10% annual discount rate is applied, such that $\delta = 0.1/365$ has been chosen on the industrial instance set. For the experimental analysis, we ran the algorithm with the objective to minimize the makespan for one hour and obtained the makespans as tabulated in Table 4.5. Given that makespan, a CP and a CIP run have been performed for one and five hours. Table 4.5 shows the NPVs and in the last column the best dual bound on the NPV which is always found by the CIP approach. For the large instances only few nodes can be explored during search which is certainly a drawback for a branch-and-bound approach. For instances GL and W, the CP approach finds slightly better primal solutions than the CIP approach. In contrast for instances Z, ZF and NZ_def, the CIP approach generates much better solutions than the CP approach.

Conclusion

We studied RCPSPDC from a CIP perspective. To model the non-linear objective function, a constraint handler for `npv`-constraints has been introduced into the CIP framework SCIP. Better heuristics, like the scatter search, genetic algorithms or local search, could be added in order to improve upon the solution qualities, and especially to generate initial feasible solutions faster.

We tested how much more computational effort is needed to solve RCPSPDC in contrast to RCPSP. The framework is able to handle instances with up to $10,000$ jobs, but for instances of that size, the quality guarantees are very weak. The computational study confirmed for RCPSPDC that IP techniques, such as the proposed continuous relaxation, are an important ingredient if cash-flows are as well positive and negative. Then, a CP approach needs a huge amount of computation time to find good or sometimes even optimal solutions. On the other hand, CP techniques perform fast on makespan objective and on RCPSPDC instances where all cash-flows are only non-positive or non-negative.

[2] The data has been generated by Chris Alford from real world projects by aggregating the original data and deleting some extra constraints such as maximum time-lags, forbidden parallel execution of jobs and minimum working rates.

4.2 Application to Labor-Constrained Scheduling Problems

In the basic version of RCPSP, each job has a constant resource demand during its processing time. There are many reasons to extend this model and to allow varying resource demands. On the practical side, the number of workers (or machines) needed in the beginning may be different from the number needed at the end. This perspective is important when a project reflects an aggregated problem in which several projects are optimized on a broad basis. This problem has been studied as *Labor-Constrained Scheduling Problem*, or *LCSP* for short, see e.g., [50, 139, 153].

In this section, we adapt the CIP framework for RCPSP by generalizing the propagation and explanation algorithms of the cumulative constraint (see Section 2.3) to the case of varying demands (Section 4.2.3). We elaborate on the hardness of such instances and present examples that illustrate the weaknesses of CP techniques and of list scheduling heuristics. In Section 4.2.4, the developed techniques are evaluated on a set of instances from [48] that have been generated from practical data sets. The SAT techniques do not reveal much improvements on the solving process, whereas a pure CP approach is able to handle medium sized instances with high efficiency and improves several known dual bounds. Good primal bounds are derived by using a MIP formulation and a list scheduling heuristic based on the LP solution values.

4.2.1 Problem description

In LCSP we are facing a variant of RCPSP, see Section 1.1, where a set of jobs $\mathcal{J}$ with processing times $p_j \in \mathbb{Z}$ for $j \in \mathcal{J}$, a set of resources $\mathcal{R}$ with capacities $R_k \in \mathbb{Z}$ for $k \in \mathcal{R}$ and a precedence graph $G = (\mathcal{J}, A)$ are given. Additionally, in this setting the resource demand of a job j per resource k varies over time in discrete time steps and is therefore given by a discrete *resource demand profile* $r_{jkt} \in \mathbb{N}_0$ for each time unit t over the processing interval, i.e., $t \in \{0, \dots, p_j - 1\}$. Outside the processing interval, the resource demand is zero. We denote by $\boldsymbol{p}$ the vector of processing times and by $\boldsymbol{r}$ the vector of resource demand profiles per job.

The goal in LCSP is to find a schedule $\boldsymbol{S}$ (an assignment of start times for each job) with minimum makespan such that the following resource constraints are satisfied:

$$\sum_{j \in \mathcal{J}: 0 \le t - S_j < p_j} r_{jk,t-S_j} \le R_k \qquad \forall\, t, \forall\, k \in \mathcal{R}.$$

Example 4.1. Figure 4.3 shows an instance consisting of three (non-dummy) jobs and a corresponding earliest start schedule.

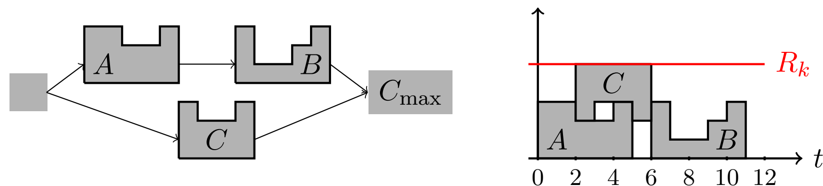


Figure 4.3: A precedence network of three jobs and an earliest start schedule.

4.2.2 Related work

LCSP has been first studied in the project PAMIPS as introduced by Kallrath and Wilson [153]. Heipcke et al. [139] describe instance sets that are obtained from two original instances at BASF SE[3] and can be downloaded from [48]. The precedence networks in these instances can be clustered into sub-networks (called *orders*) that are arranged as a chain. Additionally, there are few precedence relations between jobs from different orders. Hence, these instances have a low precedence ratio. For low capacity values (say 18) they are of disjunctive type (at least at the peaks of the resource demand profiles), and for high capacity values (say 24) of cumulative type. The classical IP formulations that introduce binary variables per possible start time yield a weak LP relaxation since resource constraints can be easily satisfied using fractional values. Such a solution intuitively means that jobs are smeared over the time horizen, i.e., some percentage is executed at the beginning, some other later. Important cutting planes like the ones from Christofides et al. [59] yield better dual bounds but are computationally more time consuming, see [50, 243]. A strengthened block-based IP formulation also yields good dual bounds but was computationally not as benificial as CP approaches, see [50, 243].

With a CP based approach [138, 139] small instances (4-6 orders, 24 jobs) can be solved efficiently including an optimality proof. For larger instances, a tabu search yields the best results: Cavalcante et al. [49] present two heuristic algorithms (asynchronous team and a tabu search) that exchange information (*parallel cooperative approach*). Currently, their heuristics contribute some of the best known solutions.

Cavalcante et al. [50] discuss LP-based heuristics and the influence of different IP formulations. In their work the different approaches are evaluated and several settings are combined. LP-based heuristic solutions (also see [218, 219, 220]) are improved via local search. The heuristics are based on α-points [123], i.e., the first point in time, at which in an LP solution the sum of the fractional time-indexed variables per job sum up to at least a value of α. Schulz and Skutella [224] show that if for each job a value α_j is chosen at random and the jobs are scheduled according to these points, one obtains a 1.693-approximation algorithm for single machine scheduling with release times and the objective to minimize the sum of weighted completion times ($1|\mathsf{r}_j|\sum w_j C_j$).The authors from [50] report that when using a list scheduling heuristic throughout branch-and-bound, the impact of stronger formulations is negligable.

4.2.3 The `labor`-constraint

To model LCSP within a constrained integer program, we introduce a `labor`-constraint that captures the propagation, relaxation and explanation techniques of each resource constraint $k \in \mathcal{R}$ imposed by LCSP:

$$\texttt{labor}(\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}_{\cdot k \cdot}, R_k) : \qquad \left\{ \boldsymbol{S} \in D(\boldsymbol{S}) \mid \sum_{j \in \mathcal{J} : S_j \leq t < S_j + p_j} r_{jk, t - S_j} \leq R_k, \forall\, t \right\}.$$

[3] BASF SE is a world-wide operating chemical company, see `www.basf.com`.

Given this constraint, a CIP model for LCSP reads as follows:

$$\begin{aligned}
\min \quad & C_{\max} & \\
\text{subject to} \quad & \texttt{precedence}(S_i, S_j, p_i) & \forall\,(i,j) \in A \\
& \texttt{labor}(\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}_{.k.}, R_k) & \forall\, k \in \mathcal{R} \\
& D(S_j) = \mathbb{N}_0 & \forall\, j \in \mathcal{J}.
\end{aligned}$$

Next, we extend the propagation and explanation algorithms known for RCPSP and discuss their applicability. Then, LP techniques and the use of heuristics are discussed.

4.2.3.1 Propagation

When considering RCPSP, the resource profile (of a solution or of the core-profile) could be represented efficiently by considering at most $2n$ different points in time, i.e., the start and end times of each job. In case of LCSP with varying demand profiles per job, the propagation procedures become more expensive, e.g., the resource profile for checking whether a solution is resource-feasible must respect each point in time since at almost each point in time a profile change may occur. Hence, checking the feasibility of a solution to LCSP is not polynomially bounded in the number of jobs. But as the resource demand profiles per job are given per unit of processing time in the input, the overall resource profile is polynomially bounded in the input but often much larger than n.

Energetic arguments (overload checking, edge-finding, time-table edge-finding or energetic reasoning) become much weaker for LCSP since a job leaves more free space. We consider the example of Figure 4.4. Though a job j may not be feasibly scheduled at est_j due to the energy requirements of the other jobs, it can be scheduled at $\text{est}_j + 1$, and the lower bound on the start time cannot be shifted into the interval. It is easy to generate a scenario where e.g., the earliest start time est_j is a valid start time but not $\text{est}_j + 1$. With the varying demands, there are $O(T^2)$ intervals to be considered in energetic reasoning, opposed to $O(n^2)$ as for RCPSP. Hence, extending the cumulative propagators must be done carefully. We restrict to the overload checking procedure of edge-finding and do not use energetic reasoning. The energy requirement of job j is given by $e_j = \sum_{\tau=0}^{p_j-1} r_{j\tau}$. Then, overload checking can be performed in $O(n \log(n))$, as for RCPSP.

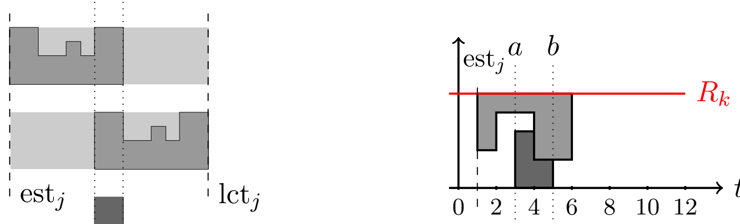


Figure 4.4: On the left: the core of a job for LCSP. On the right: a job can be scheduled at $\text{est}_j + 1$, but the totally required capacity would exceed the available volume in interval $[a, b)$ if scheduled at est_j.

Turning to time-tabling, the core of a job (if the job is not fixed) is only computed as the minimum over a set of possible demands. For the resource profile, we use an array

to store the demand at each point in time. In time-tabling, we compute the core of a job (see Figure 4.4) as a function $\gamma_j(t)$ varying over time by defining:

$$\gamma_j(t) := \begin{cases} \min_{\tau=t-\text{lst}_j,\dots,t-\text{est}_j}\{r_{j\tau}\} & t \in [\text{lst}_j, \text{ect}_j) \\ 0 & \text{otherwise.} \end{cases}$$

We remark that this formula gives the best possible estimation of a core at time t. The value τ is well-defined for the given values of $t \in [\text{lst}_j, \text{ect}_j)$. It turned out to be crucial to use such a precise definition of the core, instead of e.g. using simply the minimum demand of a job which would result in a core-profile with at most $2n$ points in time to be considered, but this does not yield enough propagation.

Explaining time-tabling propagation is hard even if a set of locally fixed jobs needs to be selected. In fact, we show that this is a strongly $\mathcal{NP}$-hard problem by reduction from 3-SAT. The problems and the reduction are introduced in the next section.

4.2.3.2 Complexity of explaining time-tabling for LCSP

We first state the problem that is used in the reduction.

Problem: **3-SAT**

Instance: A set of m clauses over boolean variables $x_1, \dots, x_n$. Each clause contains exactly three variables.

Question: Is there an assignment $x^\star \in \{0,1\}^n$ such that all clauses are satisfied?

This problem is known to be strongly $\mathcal{NP}$-complete [155].

We consider a special case in which we need to explain a lower bound change from est_j to est'_j in the context of LCSP. In this special case, the resource demand of the updated job j is constant over time and given by $r_j = R \geq 6$. These parameter values are chosen in a way that the instance is not infeasible before the update. The processing time of the job is chosen as $p_j = 3$. Hence, we need to explain at least one peak of height at least one within every interval of size three. We identify this to be a *minimum interval resource covering problem* (MIRCP).

Problem: **MIRCP**

Instance: A set $\mathcal{J}$ of n jobs fixed in non-empty intervals $[s_k, t_k)$ per job k with a varying demand $r_{kt} \in \mathbb{Z}_0$ for all t, a minimum resource requirement $h \in \mathbb{Z}^+$, a length p and a non-empty interval $[a, b)$.

Question: Is there a subset $\Omega \subseteq \mathcal{J}$ of ℓ jobs such that for every interval $I \subseteq [a, b)$ of size p there exists a $t \in I$ for which $\sum_{k\in\Omega: s_k \leq t < t_k} r_{kt} \geq h$ holds?

In MIRCP we ask to find a subset of jobs such that every interval of size p that is a sub-interval of $[a, b)$ is covered with a demand at least h. This corresponds to an explanation for a bound change from $\text{est}_j = a$ to $\text{est}'_j = b$ where $R - r_j = h - 1$ and the processing time of the updated job is given by $p_j = p$.

Before proving that finding a minimum size explanation for bound changes derived by time-tabling is strongly $\mathcal{NP}$-hard, we remark that verifying a solution for a given LCSP instance can only be performed in polynomial time, if the processing times of all jobs are either part of the input or are bounded by the number of jobs. Otherwise, we can only verify in pseudo-polynomial time, whether a solution to LCSP is feasible, hence the problem is not in $\mathcal{NP}$ in general. The same argument holds for checking whether

an explanation for the time-tabling update is valid. The reduction employed next only needs processing times that are bounded in the number of variables of the given 3-SAT instance.

The LCSP gadget we create consists of a variable side and a clause side. On the variable side, we introduce consecutively in an arbitrary order for each variable of the 3-SAT instance an interval of size three. These intervals are numbered $I_{x,1}, ..., I_{x,n}$ where the second index corresponds to the variable index in the 3-SAT instance. Let $I_{x,k} = [3(n-1), 3n)$ for $k = 1, ..., n$. The first and third point in time of each interval corresponds to the 'true' assignment of the corresponding variable and the second point in time to a 'false' assignment. On the clause side, we consecutively introduce m intervals $I_{C,1}, ..., I_{C,m}$, one for each clause, with $I_{C,k} = [3(n+k-1), 3(n+k))$ for $k = 1, ..., m$. The distance between a and b is $3 \cdot (n+m)$. We use $h = 1$ and $p = 3$, hence each interval of size three needs to be explained with at least one job.

Now, we introduce the jobs of the MIRCP instance and their intervals $[s_k, t_k)$ for $k \in \mathcal{J}$. For each variable x_k we introduce two jobs T_k and F_k, corresponding to the assignment of this variable to 'true' and 'false', respectively. We set the length of the interval $[s_k, t_k)$ to $p_k := 3 \cdot (n+m)$. For easier reading we set all processing times to $3 \cdot (n+m)$ though this length can be bounded by $3 \cdot (n-k+1+m_k)$ where m_k denotes the largest index of all clauses this variable appears in. The start point of the interval of job T_k can also be bounded by $s_k = 3 \cdot (k-1)$, which we again relax to $s_k := 0$ for simplifying the remaining indices. Still, all values are polynomially bounded in the number of variables and clauses of the 3-SAT instance. The resource demand of a job T_k is given by

$$r_{kt} = \begin{cases} 1 & t = 3 \cdot (k-1), 3 \cdot (k-1) + 2 \\ 1 & t : t \in I_{C,\ell} \wedge x_j \in C_\ell, \ell = 1, ..., m \\ 0 & \text{otherwise.} \end{cases}$$

The resource demand of a job F_k is given by

$$r_{kt} = \begin{cases} 1 & t = 3 \cdot (k-1) + 1 \\ 1 & t \in I_{C,\ell} \wedge \bar{x}_j \in C_\ell, \ell = 1, ..., m \\ 0 & \text{otherwise.} \end{cases}$$

Figure 4.5 shows the construction schematically.

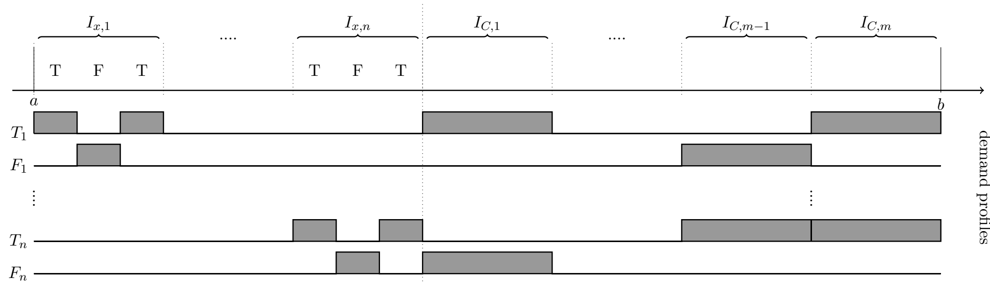


Figure 4.5: Construction of an MIRCP instance for a 3-SAT instance consisting of n variables and m constraints.

A more descriptive example is given next.

Example 4.2. We consider the following instance of 3-SAT:

$$(x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4)$$

We set $a := 0$ and obtain $b := 21$. A peak of height one must be explained in every interval of size three. Eight jobs are introduced with the depicted resource demand profiles. Turning to the task to explain a bound update, the job whose lower bound is updated from $\text{est}_j = 0$ to $\text{est}'_j = 21$ has processing time three and a resource demand of six resource units while the capacity is six.

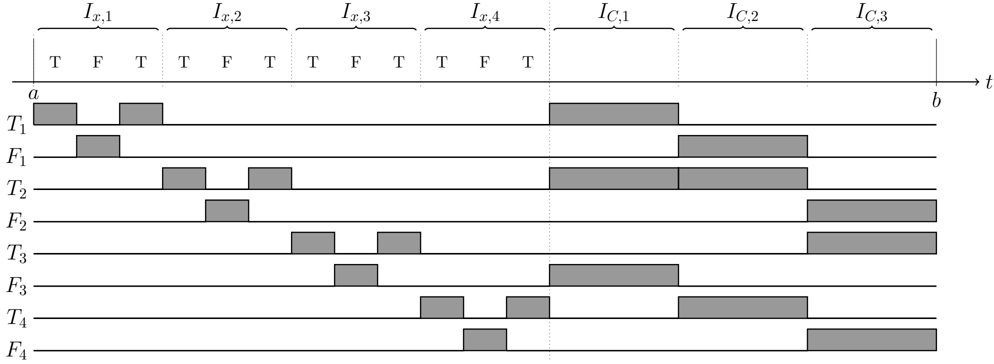


Figure 4.6: An instance of MIRCP after applying the reduction from the following instance of 3-SAT: $(x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4)$.

Now, we prove that a yes-instance for 3-SAT corresponds to a yes-instance of MIRCP and vice versa.

Theorem 4.4. *There exists a truth-assignment for the 3-SAT instance with n variables if and only if there exists a cover Ω with at most n jobs.*

Proof. Let $x^\star \in \{0,1\}^n$, where 1 corresponds to 'true' and 0 to 'false', be a satisfying truth assignment of the 3-SAT instance. We consider the corresponding MIRCP instance. Then, for each variable x_k set to true, we put job T_k into Ω and for each variable x_k set to false, we put job F_k into Ω. We prove next, that Ω covers at least one peak in each subinterval of $[a, b)$ of size three. Note that $|\Omega| = n$. For each interval I_{xk}, $k = 1, ..., n$ at least one peak is explained, as either $T_k \in \Omega$ or $F_k \in \Omega$. In sub-intervals between two intervals I_{xk} and $I_{x,k+1}$, a covered peak is at most three units ahead due to the choice of the resource demand profile. For each clause interval $I_{C,\ell}$, $\ell = 1, ..., m$, there exists a job $k \in \Omega$ with resource demand $r_{k,3\cdot(n+\ell-1)} > 0$, since each clause is satisfied. Thus, each interval is covered.

Consider the converse of the statement. Given a cover Ω with $|\Omega| = n$. The case $\Omega < n$ would not cover all intervals on the variable side and can therefore be omitted. For each $k = 1, ..., n$, exactly one of the jobs T_k or F_k is contained in Ω as from each variable interval at least one job must be contained in Ω and $|\Omega| = n$. Hence, we set the variable x_k to true if $T_k \in \Omega$ and false if $F_k \in \Omega$. as in each clause interval a peak with value at least one is given, at least one of the jobs contributes a non-zero resource demand to that interval and thus the corresponding clause is satisfied. □

A similar complexity result has been obtained to explain bound changes for time-tabling in an RCPSP setting. There, our reduction presented in [34] introduces weights

in the objective function that can get arbitrarily large while Chrobak et al. [60] introduce demands that can get arbitrarily large. In our reduction for LCSP only the processing times get arbitrarily large and in this case finding n out of $2n$ jobs that cover each point in time is strongly $\mathcal{NP}$-complete.

4.2.3.3 IP formulation and relaxation

Using a global `labor`-constraint, we are able to use different kinds of IP relaxations for LCSP. As for RCPSP, we complement the pure CP search with the classical IP formulation from Pritsker et al. [209] as presented in Section 1.3.1, that in case of LCSP reads as follows:

$$\text{minimize} \quad C_{\max}$$

subject to

$$S_i + p_i \leq S_j \qquad \forall\,(i,j) \in A$$

$$\sum_{t=\text{est}_j}^{\text{lst}_j} t \cdot x_{jt} = S_j \qquad \forall\, j \in \mathcal{J}$$

$$\sum_{t=\text{est}_j}^{\text{lst}_j} x_{jt} = 1 \qquad \forall\, j \in \mathcal{J}$$

$$\sum_{j \in \mathcal{J}} \sum_{\tau=\max\{0,t-p_j+1\}}^{t} r_{j,k,t-\tau} \cdot x_{j\tau} \leq R_k \qquad \forall\, k \in \mathcal{R}, t$$

$$\text{est}_j \leq S_j \leq \text{lst}_j \quad \forall\, j \in \mathcal{J}, \quad x_{jt} \in \{0,1\} \quad \forall\, j \in \mathcal{J}, t.$$

Comparing this formulation with the one for RCPSP, we see that only demand coefficients change; but still each inequality of the resource constraints corresponds to a knapsack inequality. In the aggregated formulation (using z_{jt} variables that indicate whether job j has been started until t), negative coefficients may occur. The column generation based IP formulation by Mingozzi et al. [186] cannot be applied due the varying demands. Hence, we use the formulation presented above.

Precedence inequalities As mentioned by Cavalcante et al. [50], in order to obtain better dual bounds from the LP relaxation, strong precedence inequalities can be used. Because there are many of them at hand (number of time units times the number of precedence relations), these should be separated throughout branch-and-bound search. These cuts have been introduced by Christofides et al. [59] and with some reformulations, we can express them in the following ways:

1. Set-packing formulation: $\forall\; t, (i,j) \in E$:

$$\sum_{\tau=t}^{\text{lst}_i} x_{i\tau} + \sum_{\tau=\text{est}_j}^{t+p_i-1} x_{j\tau} \leq 1. \tag{4.6}$$

2. Logicor formulation: $\forall\ t, (i,j) \in E$:

$$\sum_{\tau=\mathrm{est}_i}^{t-1} x_{i\tau} + \sum_{\tau=t+p_i}^{\mathrm{lst}_j} x_{j\tau} \geq 1 \tag{4.7}$$

3. Shifting formulation: $\forall\ t, (i,j) \in E$:

$$\sum_{\tau=t}^{\mathrm{lst}_i} x_{i\tau} \leq \sum_{\tau=t+p_i}^{\mathrm{lst}_j} x_{j\tau}. \tag{4.8}$$

These inequalities are illustrated in Figure 4.7. By a simple calculation, we next show that all these formulations are equivalent.

Lemma 4.5. *The set-packing* (4.6)*, logicor* (4.7) *and shifting* (4.8) *formulation are equivalent.*

Proof. Consider an assignment of the variables $[x_{it}]_{t=\mathrm{est}_i,\dots,\mathrm{lst}_i}$ and $[x_{jt}]_{t=\mathrm{est}_j,\dots,\mathrm{lst}_j}$ satisfying (i) $\sum_{t=\mathrm{est}_i}^{\mathrm{lst}_i} x_{it} = 1$ and (ii) $\sum_{t=\mathrm{est}_j}^{\mathrm{lst}_j} x_{jt} = 1$. Then, we calculate for a point in time t:

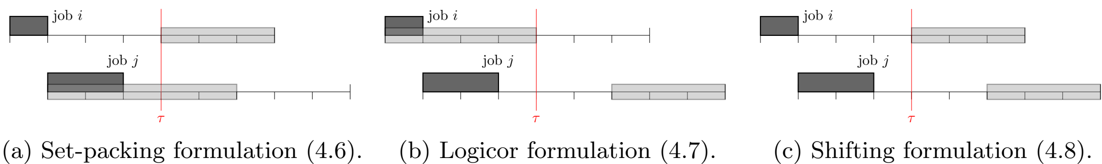
$$\begin{aligned}
& \sum_{\tau=t}^{\mathrm{lst}_i} x_{i\tau} + \sum_{\tau=\mathrm{est}_j}^{t+p_i-1} x_{j\tau} \leq 1 \\
\overset{(ii)}{\Leftrightarrow}\quad & \sum_{\tau=t}^{\mathrm{lst}_i} x_{i\tau} + \left(1 - \sum_{\tau=t+p_i}^{\mathrm{lst}_j} x_{j\tau}\right) \leq 1 \\
\Leftrightarrow\quad & \sum_{\tau=t}^{\mathrm{lst}_i} x_{i\tau} \leq \sum_{\tau=t+p_i}^{\mathrm{lst}_j} x_{j\tau} \\
\overset{(i)}{\Leftrightarrow}\quad & \left(1 - \sum_{\tau=\mathrm{est}_i}^{t-1} x_{i\tau}\right) \leq \sum_{\tau=t+p_i}^{\mathrm{lst}_j} x_{j\tau} \\
\Leftrightarrow\quad & \sum_{\tau=\mathrm{est}_i}^{t-1} x_{i\tau} + \sum_{\tau=t+p_i}^{\mathrm{lst}_j} x_{j\tau} \geq 1
\end{aligned}$$

□

The set-packing formulation is the one presented by Christofides et al. [59]. Though these formulations are equivalent, when separated throughout branch-and-bound search we will see in the computational study that they have a different impact on the solving process.

4.2.3.4 Heuristics

In order to generate good primal solutions, we apply a list scheduling heuristic after each propagation round and during presolving. Here, jobs are sorted according to their start times in an LP or pseudo-solution, or by weighted start, completion and processing times. As weights we choose $\alpha, \beta \in [0,1]$ and set $S_j := \alpha\, \mathrm{est}_j + (1-\alpha) \cdot \mathrm{lst}_j + \beta \cdot p_j$. Observe that



(a) Set-packing formulation (4.6). (b) Logicor formulation (4.7). (c) Shifting formulation (4.8).

Figure 4.7: Formulations of the strong precedence cuts from left to right. The involved binary variables of jobs i and j are marked in light grey.

if $\alpha = 1$ and $\beta = 0$, an earliest start schedule is obtained, whereas for $\alpha = 0$ and $\beta = 1$ an earliest deadline first strategy is used.

For heuristics we observe that, given an optimal solution for RCPSP, we can sort the jobs by their earliest start times and by inserting them in that order we again get an optimal solution. Left-Shifts and right-shifts would yield solutions that are at least as good as before. This no longer holds for LCSP as Figure 4.8 shows. Hence, in a bi-directional scheduling scheme the solution may get worse.

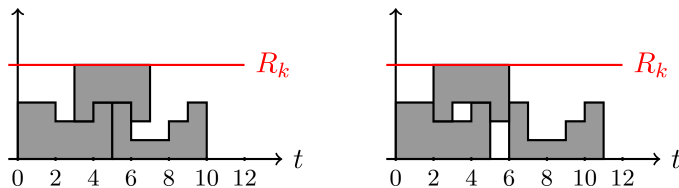


Figure 4.8: On the left, a schedule with minimum makespan is given for the LCSP instance from Figure 4.3. Scheduling all jobs in the order of the start times yields no optimal schedule as for RCPSP.

With these refinements, the computational study is carried out in the next section.

4.2.4 Computational study

We carry out a computational study on the PAMIPS instances from [48].

We present elaborate results on combinations of CP, IP and SAT techniques for LCSP, combined with a progressive or destructive search, using only inference branching or additionally reliable pseudocost branching on all variables or restricted to the integer variables. Then, we will see that separating precedence inequalities throughout search is able to reduce the number of nodes needed, but increases the running time considerably. In particular, we will see that among the different formulations of the precedence inequalities, the shifting formulation performs best. After that, numerical results indicate that it pays off to use the rather costly list scheduling heuristic throughout search. Using the best CP approach and the best settings for a CP-IP-hybrid we will see that we are able to close one more of the original PAMIPS instances and improve six lower bounds. Eleven primal bounds are better than those of other exact approaches but not competitive at all with tabu search procedures.

4.2.4.1 Instances

In each instance from the set [48], a number of orders is given. The precedence relations of jobs in one order correspond to a chain, hence there are no resource conflicts between jobs of the same order. There are few precedence relations between jobs of different orders. The capacity is set to 18. The resource demand profile of a job varies widely. It is on average six, while few peaks have a capacity of 18. Hence, no other job is allowed to be scheduled in parallel to such a peak. To summarize, the instances are of low precedence ratio and rather cumulative. The few peaks that need the whole capacity induce further disjunctions between jobs.

The instances are grouped according to the number of jobs, containing (4 orders) 21-27, (6 orders) 41-44, (8 orders) 63-65 and up to 109 jobs (12 orders). All instances are equipped with a resource capacity of 18. Further experiments are made where the capacity is changed to 21 and 24 on the smaller instances containing 4-8 orders. These will be used to compare the different settings, before a collection of best of CP, IP and CIP settings are compared on the whole instance set.

The best known solutions are reported in [49, 50] and are shown in Table 4.6. The differences between the lower and upper bounds indicate that instances which contain at most four orders can be solved to optimality. Hence, for our initial experiments in which we fine-tune the parameter settings, we use a smaller test set, denoted by 'setS', consisting of all instances with at most 6 orders. From these instances we generate additional ones by using capacity values of 18, 21 and 24. These capacity values are also recommended by the instance providers, while using higher capacity values, such instances become easier to solve as they become less disjunctive – the highest demand value is 18 and most demand values are no larger than six.

For the instances from set setS, we invoke a time limit of 600 sec. The original instances are of huge size in the time-indexed formulation resulting in long LP solving times. That's why, we invoke a time limit of one hour in the final experiments.

Instance	Primal	Dual	CP-Primal	Instance	Primal	Dual	CP-Primal
Ins3o7jA	10	10		Ins8o63jC	294	271	344
Ins4o21jA	82	82	82	Ins8o65jA	403	342	445
Ins4o23jA	58	58	58	Ins8o65jB	382	315	411
Ins4o24jA	68	68	68	Ins10o84jA	634	394	730
Ins4o24jB	72	72	72	Ins10o84jB	550	355	616
Ins4o27jA	67	67	67	Ins10o85jA	783	671	912
Ins6o41jA	140	109	152	Ins10o87jA	581	377	610
Ins6o41jB	110	102	110	Ins10o88jA	450	–	473
Ins6o41jC	126	110	134	Ins10o100jA	1467	830	1587
Ins6o44jA	117	98	122	Ins10o102jA	1155	878	1239
Ins6o44jB	137	124	149	Ins10o106jA	1087	578	1166
Ins8o63jA	259	187	281	Ins12o108jA	1271	838	1412
Ins8o63jB	314	239	344	Ins12o109jA	1324	980	1476

Table 4.6: Listing of the best known primal solutions from approaches from the literature including heuristics such as tabu search ('Primal'), and without these heuristics: best dual ('Dual') and best CP-based primal bound ('CP-Primal').

4.2.4.2 Numerical results

Combinations of CP, IP and SAT with destructive or progressive search Now, we evaluate the impact of using a standalone inference branching rule, compared to combining it with either a destructive search, a progressive search or spending half the running time on a progressive search and afterwards on a destructive search. After branching on the makespan variable has been performed, the inference branching rule is used in all settings.

These branching rules are combined with four possible combinations of CP, IP and SAT approaches. The abbreviations for the settings are given in Table 4.7.

cp	only propagation	I	inference branching
cpsat	cp + conflict analysis	D	destructive search
cpip	cp + IP relaxation	P	progressive search
cip	cpip + conflict analysis	C	half progressive, then destructive search

Table 4.7: Abbreviations for the settings.

Table 4.8: Comparison of CP, IP and SAT techniques combined with a destructive or progressive search.

setting	nopt	bprimal	bdual	gap	avtime	avnodes
setS					(allopt: 15 instances)	
cpsatD	15	15	15	10.24	29.93	538618.40
cpsatP	15	16	15	11.16	27.72	385098.00
cpsatC	15	16	17	9.39	29.19	389990.33
cpsatI	16	17	16	10.87	32.75	638505.33
cpD	**17**	17	**24**	8.38	3.73	37782.87
cpP	**17**	17	19	9.82	3.85	39709.47
cpC	**17**	17	**23**	8.21	3.85	39709.47
cpI	16	18	17	9.65	7.21	98574.60
cipD	15	15	16	9.18	15.50	35888.67
cipP	15	15	15	11.97	12.91	27999.33
cipC	15	16	16	8.74	12.91	27999.33
cipI	15	18	15	14.08	22.59	45599.20
cpipD	15	15	16	9.56	19.11	48806.33
cpipP	15	15	16	12.15	14.65	32847.20
cpipC	16	**19**	16	8.68	14.65	32847.20
cpipI	15	**19**	15	13.83	23.13	45260.73

We will first compare the settings between CP, CP-IP, CP-SAT and CIP approaches and then turn to the impact of branching on the makespan variable.

From Table 4.8 we see that a pure CP approach finds for 17 instances an optimal solution, as indicated in column 'nopt'. In contrast, the other approaches are only able to find and prove optimality of 15 or 16 solutions. In particular, using a CP approach finds most of the best dual bounds (on 24 of 27 instances), whereas a CP-IP hybrid is able to find 19 times a best primal bound, which is only slightly better than with the other approaches. The final gap is smallest when using a CP approach, 8% in contrast to up to 14% on average.

Considering the 15 instances that are solved to optimality by all settings (column 'allopt'), we see that a pure CP approach needs less than half the running time compared to CP-SAT and CP-IP approaches. In contrast due to the better dual bounds, a CIP approach needs the fewest number of nodes on average. Often, these instances are solved

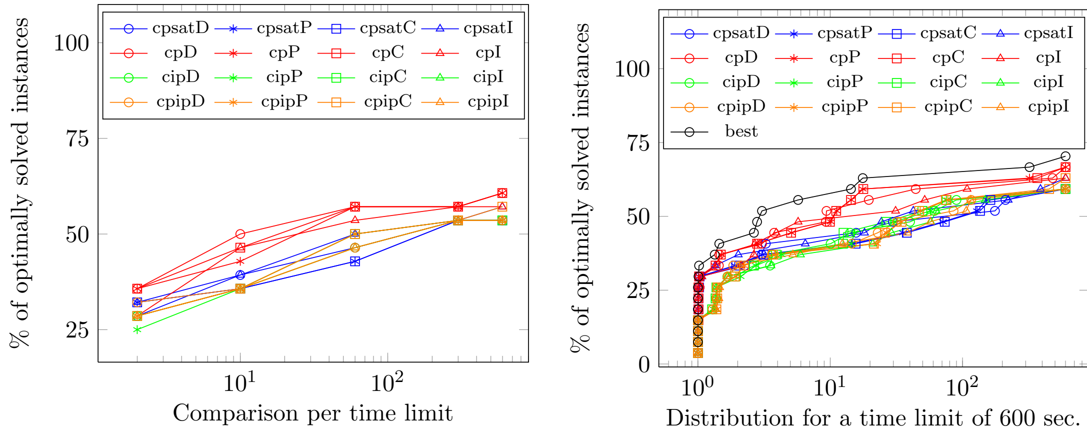


Figure 4.9: Number of optimally solved instances from setS.

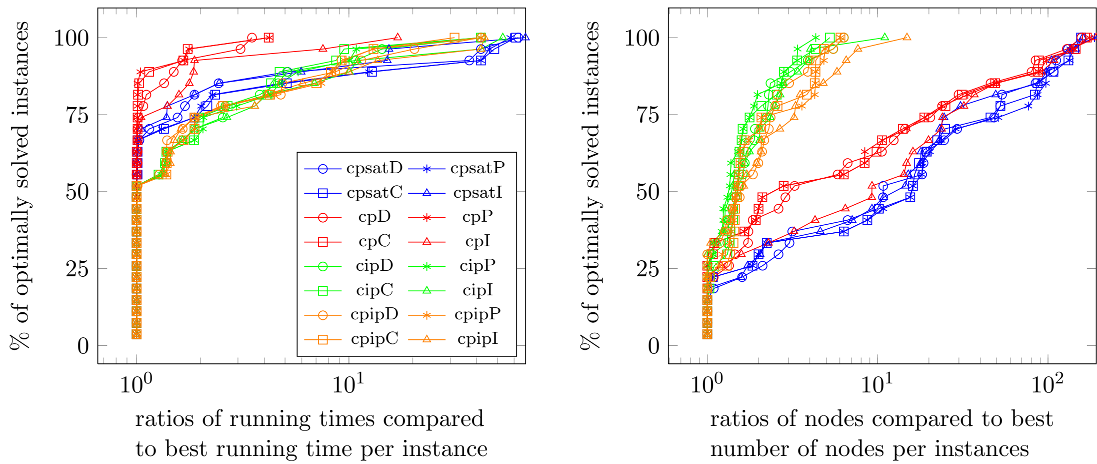


Figure 4.10: Ratios of nodes and running times given for all optimally solved instances from setS.

within less than half the running time (300 sec.), hence, there is no difference between the values reported on progressive search and the combination of progressive and destructive search. In Figure 4.10 distribution functions over the ratios computed as number of nodes (time) divided by the fewest number of nodes (lowest running time) are given. We see that a CP-IP-SAT hybrid performs best with respect to that measure, whatever branching strategy for the makespan variable is chosen. But the running times are much worse. In contrast, a CP search performs best if the measure is the ratio of the running times, whereas it needs many more nodes in contrast to the settings which involve IP. Interestingly, using conflict analysis on top of the CP solver does not pay off in total, it performs worse with respect to the number of nodes and the running time than the CP search. This might be due to the fact that the vsids are not much reusable throughout search as we collect vsids per variable and not per value in SCIP. This can also be observed in setting 'cpsatD' where many more nodes are needed than in a progressive search. It gets hard to find the optimal solution when branching according to the vsids is performed with a destructive search on top as the search is guided to infeasible spaces.

Now, we turn to the different branching schemes that can be applied to the makespan variable. In all settings, except for 'cpsatD', we see from column 'bdual' in Table 4.8 that using a destructive search or a combination of progressive and destructive search leads to the highest number of best dual bounds obtained. In contrast, using inference branching we obtain the highest number of best primal bounds. Using a combination of progressive and destructive search leads to the lowest average gaps as column 'gap' reveals. The progressive search as well as a pure inference branching are well suited when looking for good primal bounds, whereas using the destructive search rejects unpromising low makespan. Hence, a combination of both is well suited.

Study of branching rules in CIP Now, we evaluate the importance of the branching rules on a CIP approach. Therefore, we test the following settings that are denoted by "brX(a/b)". For $X = 0$ we use the default branching rules of SCIP, i.e., hybrid branching using reliable pseudocost branching on all variables (integer and binary). For $X = 1$ we restrict this rule to branching decisions based on integer variables, whereas for $X = 2$ we only use inference branching. More scheduling specific branching schemes are applied for $X = 3$, where we branch on the variable with largest float given by $\text{lst}_j - \text{est}_j$, splitting the domain into almost equal halves and for $X = 4$ we use a serial SGS branching scheme where the first unfixed variable is fixed to its lower bound or post-poned by one unit of time. Additionally, each setting gets the abbreviation 'b' if a destructive search is used and 'a' otherwise. In all settings, the list scheduling heuristic is used throughout search.

Finally, as changing combinations of CP, SAT and IP may be useful, we apply with setting "br5" a combined branching strategy in which 1/4 of the time limit is spent in a pure CP search, and after that 1/4 of the time limit is spent on a CPSAT-based destructive search and finally the remaining half of the running time the CIP approach is used. This rule summarizes our experiences as it finds good solutions fast and works strongest on the dual bound.

Table 4.9: Comparison of different branching schemes in a CIP framework for LCSP.

setting	nopt	bprimal	bdual	gap	avtime	avnodes
setS (27 instances)					(allopt: 10 instances)	
br0a	16	19	16	12.93	2.08	1000.7
br0b	16	**22**	21	**6.59**	1.87	742.5
br1a	15	17	15	11.67	1.96	1783.8
br1b	16	17	**27**	6.68	1.72	1166.8
br2a	15	17	15	13.90	1.93	1643.7
br2b	15	17	19	7.26	1.77	1337.0
br3a	15	15	15	13.97	**1.50**	**709.2**
br3b	15	18	19	7.26	1.60	1029.9
br4a	10	15	10	18.64	15.03	34045.7
br4b	10	15	10	12.18	13.43	36202.0
br5	16	**22**	16	9.33	1.29	2583.1

Within a CIP search, the serial branching scheme finds the fewest number of optimal solutions (only 10 in contrast to up to 16) which is mainly due to the low number of best dual bounds which are also 10. Turning to the reliable pseudocost based branching rules, we see that allowing branching on all variables ('br0b') finds the highest number of best primal bounds. This is due to the fact that we run a list scheduling heuristic that uses the LP solution values as ordering of the variables but is able to schedule the jobs according to the global bounds. In contrast, branching only on the integer

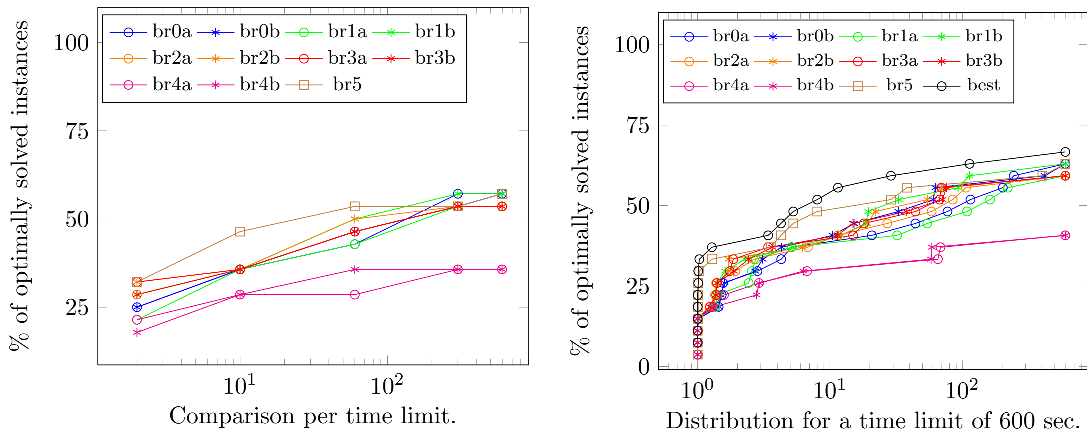


Figure 4.11: Number of optimally solved instances.

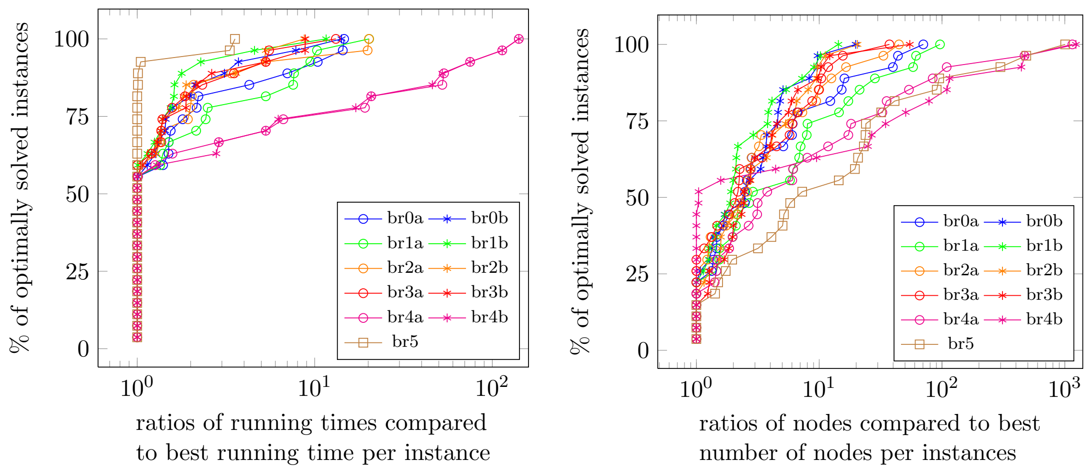


Figure 4.12: Ratios of nodes and running times given for all optimally solved instances.

variables within a destructive search ('br1b') yields the highest number of obtained dual bounds and the lowest average final gap. This branching rule splits the search spaces more eagerly than is done by branching on the binary variables and seems therefore to be better to prune unpromising nodes via propagation. Using pure inference branching ('br2a/b') is not competitive in terms of neither primal nor dual bounds.

From all the settings, we see that using a destructive search reduces the average final gap by about one-half, which results in a gap of less than 6% and increases the number of times the best dual bound could be found. On all inference and reliable pseudocost-based branching schemes, a destructive search reduces the average number of nodes needed. This is not the case if we use the median or serial branching scheme. This may be due to the conflicts learned throughout search and the vsids. Considering only the 10 instances that are solved to optimality by all settings, the median branching rule is competitive with the SCIP related branching rules in terms of running time and nodes, it is even faster on average on those instances.

The branching rule 'br5', the combination of CP, SAT and IP, shows that in particular on those instances that can be efficiently solved, the CP search in the beginning performs

fastest compared to the CIP approaches as revealed by Figure 4.9. The lower number of best dual bounds found here can be explained by less running time that is spent on using a destructive search.

Impact of precedence cuts Next, we study the impact of additional precedence inequalities. We use the following settings: In "prec0" no precedence cuts are separated, whereas precedence inequalities in the form of set-packing, logicor and shifting are checked in the settings "prec1", "prec2" and "prec3". Setting "prec4" separates all violated inequalities per separation round. Note that this way three equivalent inequalities are separated. Interestingly, this has an impact on the solution process.

Table 4.10: Impact of precendece inequalities on the solving process.

setting	nopt	bprimal	bdual	gap	avtime	avnodes
setS (27 instances)					(allopt: 15 instances)	
prec0	16	**25**	**26**	7.09	15.02	30276.47
prec1	15	18	19	8.23	23.02	31167.80
prec2	15	18	19	8.08	18.97	28581.53
prec3	15	17	18	8.49	17.28	24457.47
prec4	15	18	18	8.36	26.88	34682.80

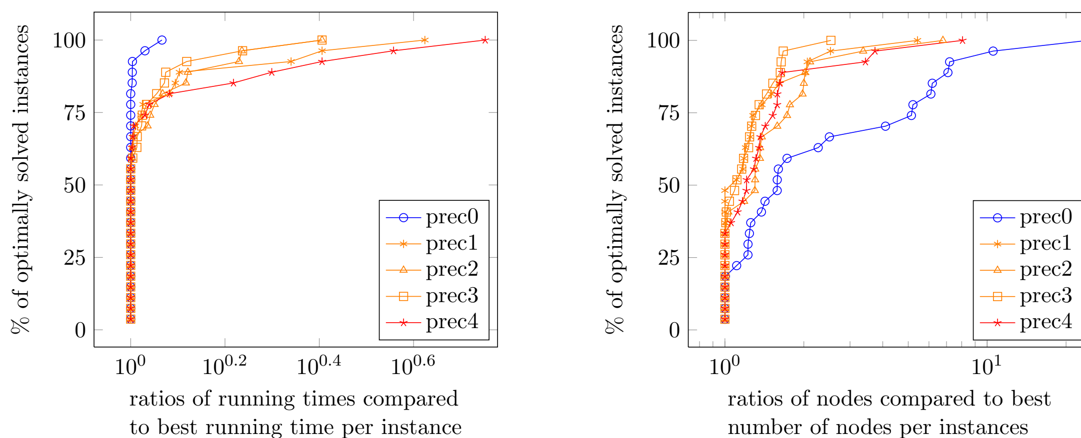


Figure 4.13: Ratios of nodes and running times given for all optimally solved instances for different formulations of the precedence inequalities.

From Table 4.10 we observe that the setting "prec0" that does not separate any precedence inequalities solves one more instance and yields the highest number of best primal and best dual bounds, except for two instances. Though we see that adding the precedence inequalities improves the dual bound of the root node, the final gap is about 1% better if no such inequalities are added. In particular, column 'gapRootDual' calculates the difference of the root dual bound to the best dual bound found by all settings. Hence, the dual bound of the root can be improved between 9.13% and 9.54% on average. Column 'gapRoot' indicates the average gap in the root node for each setting, which compared to the column 'gapRootDual' shows that gap improvements in the root node are achieved by a slightly tighter relaxation if precedence inequalities are added rather than by better primal solutions. In total the gain in the dual bound by the

precedence inequalities is rather moderate. Observe that this also stems from the fact that by using domain propagation the lower bound on the makespan is already tightened and cuts may not be that effective anymore. Hence, the influence of these inequalities decreases.

Considering the average running time and number of nodes on all instances that are solved to optimality by all solvers, again, not adding precedence inequalities is fastest on average, while adding all inequalities is slowest. Among all these settings, separating precedence inequalities in the shifting formulation leads to the fewest nodes. This is also revealed by Figure 4.13 where the distributiuon of the ratios of running times and nodes per instance are given. We explain this as follows: Using the shifting formulation some binary variables occur more frequently. These variables correspond to start times not before the current start time. Considering an LP solution, these variables are more likely to be added into the basis. Observe that all start time variables have a zero objective coefficient and can be slided in their time window as long as these are not on a critical path. Another reason why the shifting formulation leads to improved running times is that we allow the inference branching rule to perform branching on the fractional binary variables, as this turned out to be computationally better. Hence, the more often a variable is involved in the LP, it becomes more likely to enter into the LP basis with fractional value and may therefore be selected more often as a branching variable.

Impact of heuristics As setting up the resource profile for large instances is costly, intuitively, list scheduling based heuristics should not be executed to often. We show that in a CP as well as in a CIP solver executing the heuristic in each node of the search tree is better than to restrict the heuristic to the root node or to a certain depth level. We remark that experiments in which the heuristic is executed only each second, third,... node have been performed and show a slightly better performance than executing it in each node. But on some instances the solving time got much worse because better solutions are found late.

Hence, in a CP setting we compare the results if the heuristic is only executed in the root node ('heurCP0'), until tree depth 10 ('heurCP1') or in each node ('heurCP2'). Looking at the LP-based CIP approach, we study the setting 'heurCIP0' in which the heuristic is only executed in the root node and 'heurCP1' where the heuristic is executed in each node. The results are given in Table 4.11 and Figures 4.14.

Table 4.11: Impact of using a heuristic only in root node ('heurCP0', 'heurCIP0'), until tree depth 10 ('heurCP1') or in each node ('heurCP2', 'heurCIP1').

setting	nopt	bprimal	bdual	timeProp	timeHeur	gap	avtime	avnodes
setS (27 instances)							(allopt: 17 instances)	
heurCP0	17	17	26	206.7	0.51	10.13	22.7	224866.82
heurCP1	17	17	26	207.83	0.57	9.82	22.99	226306.18
heurCP2	17	**27**	**27**	175.02	35.17	**7.62**	21.32	**185005.18**
setS (27 instances)							(allopt: 15 instances)	
heurCIP0	16	16	22	44.25	0.18	9.39	69.65	128234.73
heurCIP1	15	**26**	23	40.84	8.8	**8.04**	58.36	**92104.13**

Interestingly, using the 'time consuming' heuristic pays off in total as this also reduces the average time spent in propagation due to better upper bounds on the variables which helps to find globally tighter bounds.

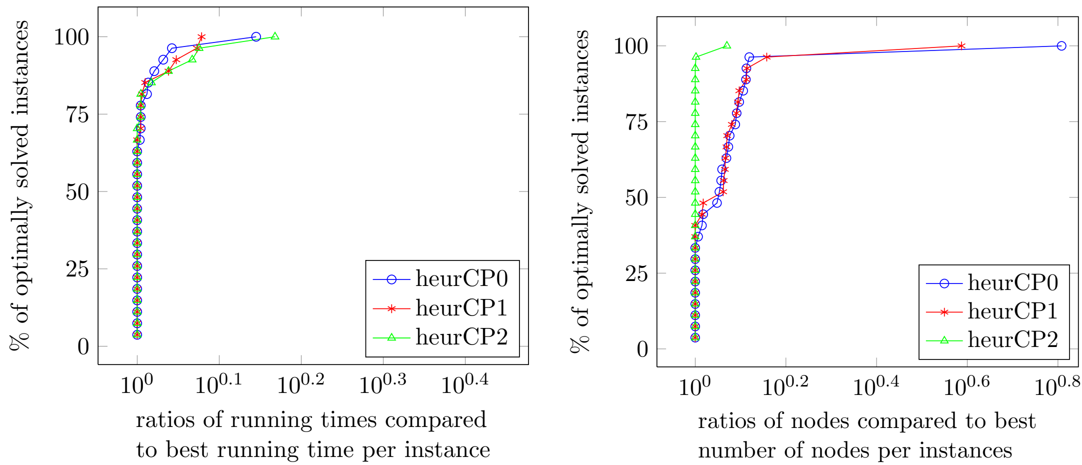


Figure 4.14: Ratios of nodes and running times given for all optimally solved instances.

First, we consider the CP settings in detail. With the setting heurCP2, we find on all instances the best upper bound and need slightly less running time and about 20% less nodes on average for those instances that could be solved to optimality by all settings. In particular, Figure 4.14 shows that heurCP2 needs on more than 95% of the instances the fewest nodes, whereas the distribution of the running times are slightly in favor to use setting heurCP0. On all instances, the final gap can be decreased to 7.62% on average in setting heurCP2, which is a decrease by more than 20%.

Considering the CIP settings, we see that heurCIP1 uses one instance less than heurCIP0 – this can be undone by increasing the time limit by 20 sec. Using the heuristic throughout search leads to the best primal bounds and decreases the final gap by about 15% on average. On all instances that could be solved to optimality by both settings, the average running time decreases from about 70 sec. to 58 sec., while the average number of nodes decreases from about 128,000 to 92,000. We omit figures concerning nodes and running time here – these values are always in favor of using the heuristic.

4.2.4.3 Results on the PAMIPS instances

Given the former fine-tuning of parameters, we finally evaluate the outcome on the complete set of PAMIPS instances from [48]. We initially performed experiments with a CP, CP-IP, CP-SAT and a CIP hybrid with the best settings obtained so far and a combination of all. As it turned out, using conflict analysis slowed down the solving process too much, hence we omit these results here and only show the best values obtained by a CP ('cp') or a CP-IP-hybrid ('cpip') where the list scheduling based heuristic is used in each node of the search tree and no cuts based on stronger precedence inequalities are separated.

Table 4.12 shows the best primal and dual bounds obtained so far compared to the CP and CP-IP approaches presented here. The best primal bound on the PAMIPS instances is found by tabu search or other heuristics as already described in Table 4.6. Instances with more than four orders could not be solved to optimality. The last four columns show the 'best primal' bound obtained by a CP, a CP-IP or by both solvers as indicated in column 'solver' and the 'best dual' bound and which 'solver' obtained this. If as well

Instance	Primal	Dual	Primal(CP)	best primal	solver	best dual	solver
Ins3o7jA	10	10	–	10	all	10	all
Ins4o21jA	82	82	82	82	all	82	all
Ins4o23jA	58	58	58	58	all	58	all
Ins4o24jA	68	68	68	68	all	68	all
Ins4o24jB	72	72	72	72	all	72	all
Ins4o27jA	67	67	67	67	all	67	all
Ins6o41jA	140	109	152	148⋆	cpip	116	cp
Ins6o41jB	110	102	110	**109**	all	**109**	all
Ins6o41jC	126	110	134	129⋆	cpip	105	cp
Ins6o44jA	117	98	122	119⋆	cpip	**105**	cp
Ins6o44jB	137	124	149	141⋆	cpip	**126**	cp
Ins8o63jA	259	187	281	275⋆	cpip	**207**	cp
Ins8o63jB	314	239	344	346	cpip	225	cp
Ins8o63jC	294	271	344	316⋆	cpip	262	cp
Ins8o65jA	403	342	445	412⋆	cpip	**349**	cp
Ins8o65jB	382	315	411	413	cpip	264	cp
Ins10o84jA	634	394	730	686⋆	cp	338	cp
Ins10o84jB	550	355	616	624	cp	280	cpip
Ins10o85jA	783	671	912	881⋆	cpip	562	cp
Ins10o87jA	581	377	610	620	cp	286	cpip
Ins10o88jA	450	–	473	474	cp	**362**	all
Ins10o100jA	1467	830	1587	1576⋆	all	487	cp
Ins10o102jA	1155	878	1239	1304	cp	631	cp
Ins10o106jA	1087	578	1166	1201	cp	490	cp
Ins12o108jA	1271	838	1412	1397⋆	cp	569	cp
Ins12o109jA	1324	980	1476	1477	cpip	910	cp

Table 4.12: Comparison of best know primal and dual values from the literature (columns 1-3) with CP and CP-IP approaches (columns 4-7). **Bold** values show improved bounds. Entries are marked by a '⋆' if our approach is better than the best CP approach from literature.

a CP as well as a CP-IP-hybrid found the best value, this is indicated by writing 'all'.

With our approach we are able to close one more instance containing six orders, i.e., Ins6o41jB with objective value 109. We are not able to find better solutions on the larger instances but we obtain eleven times a better primal bound compared to the former CP approaches, see [49] and [50]. The corresponding values are marked by a '⋆' in Table 4.12. Furthermore, we are able to improve six dual bounds (marked in bold).

Most of our best primal bounds are obtained using the CP-IP hybrid, whereas most of our best dual bounds are obtained by the CP approach due to the larger number of nodes that are explored. In total, the gaps between the best dual and primal bound on the large instances remain huge.

Conclusion

In this section we extended the CIP framework for RCPSP and in particular the time-tabling algorithm of the cumulative constraint to LCSP.

LCSP is intrinsically harder than RCPSP as RCPSP is included in case the demands are not varying. Checking feasibility of a solution can only be done in time depending on the processing times as the resource demands are given per unit of processing time. Time-tabling and explaining time-tabling become much harder as shown by reduction from 3-SAT where the processing times are polynomially bounded in the number of jobs.

As seen in the numerical results, a CP approach is fastest in finding good solutions on the small instances (up to 4 orders), whereas a CIP approach finds the highest number of best primal bounds on the larger instances (6-8 orders). Combining the CP approach using the inference branching rule with a destructive branching scheme yields good dual bounds. In total, we are able to close one instance with six orders and improve six dual bounds. The approach is not competitive with the elaborate tabu search or other heuristics in finding good primal solutions, but our exact approach is eleven times better in deriving a primal bound than exact approaches from the literature.

In total, it did not pay off in terms of running time to use conflict analysis on these instances because the conflicts are not reusable, i.e., they do not propagate that often. In contrast, the inference branching rule already detects promising branching candidates that induce further propagation.

Chapter 5

Turnaround Scheduling

In this chapter, we study the *Turnaround Scheduling Problem.* This problem is motivated by an industrial application from chemical manufacturing which we studied in cooperation with the consultancy company T.A. Cook Consultants[1].

This problem extends RCPSP in many ways: Jobs can be executed in different modes which are determined by the number of workers assigned to a job. Availability of workers is restricted to predetermined working shifts due to sleeping and break times. The working shifts depend heavily on the group of workers (craftsmen have long day shifts, whereas observer need to watch the progress from time to time and may only be available in the morning or afternoon). The main scheduling decision that needs to be made in turnaround scheduling is to determine the number of external workers hired over the planning horizon. These are the main cost drivers.

Contribution

Real-world instances for Turnaround Scheduling problems are of large scale, hence, heuristics are the main focus in practice. We will present heuristics that handle multi-mode jobs, resource availabilities and focus on minimizing the resource costs. To evaluate these heuristics, we generate classes of instances and compare the solutions to optimal solutions that will be generated by a time-indexed MIP.

To improve the lower bound, we apply a Dantzig-Wolfe decomposition to the plain MIP formulation in which pricing problems per working shift need to be solved. Getting an exponential number of variables, we propose a branch-price-and-cut algorithm to solve this model. We are able to compute optimal schedules for instances with up to 50 jobs, which is a large number in this area. In contrast, CPLEX on the plain MIP is only able to solve 25% of the instances. A detailed comparison demonstrates the gain of much tighter dual bounds obtained from Dantzig-Wolfe decomposition over the plain MIP.

Furthermore, we show how to integrate valid precedence inequalities from the original model into the reformulated master problem. These inequalities need to be separated carefully to not slow down the solving process.

Altogether, we present a novel successful application for the power of applying Dantzig-Wolfe decomposition to solve large-scale project scheduling problems with difficult characteristics and a non-standard objective function in acceptable time. The results show that even if the model involves $\mathcal{NP}$-hard pricing problems, the gain of

[1] See `de.tacook.com` for further information.

tighter dual bounds sacrifices such an approach.

Outline

In Section 5.1 the problem is described and related work is presented. Heuristics able to handle real-world instances are presented in Section 5.2. Section 5.3 deals with the obstacles of plain MIP formulations and presents the Dantzig-Wolfe decomposition. The new model is solved in a branch-price-and-cut framework as described in Section 5.3.2. The computational study in Section 5.4 emphasizes the gain of stronger dual bounds obtained from the decomposition.

5.1 Problem description and related work

For the inspection and renewal of parts in chemical plants, these are shut down and disassembled, work is done, and plants are finally rebuilt. As a consequence, a partial ordering of jobs needs to be done. Jobs are *multi-mode*, which means, they can be sped-up to a certain extent by investing in more workers. The time horizon and the number of workers hired for each job determine production downtime and working costs. That is, we have a time-cost tradeoff, where a possible solution approach is to binary search over the time horizon. For a fixed time horizon—which we assume here—the problem turns into a resource leveling problem, i.e., we need to decide how many workers per resource are hired over the time horizon.

Workers, or more generally renewable resources, are of different specialization. Each resource is associated with *availability periods* that can be thought of as working shifts. We assume that the granularity of planning is so fine that each job needs (possibly several units of) exactly one resource. In particular, the processing time of a job depends on the number of resource units that are assigned to that job. Here, we assume that the actual work increases the more resource units of one resource are assigned due to, e.g., communication overhead. Observe that a model with multiple resources per job must encode how the processing time of a job is shortened depending on how many resource units of which resource are increased. In the worst case, planners would need to encode all possible combinations of resource assignments to a job. E.g., our industrial partners use MS Project™ software which calculates the processing time by dividing the amount of work by the number of allocated resources of exactly one resource. We follow a one-resource-per-job policy in which the need of multiple resources can be modeled via generalized precedence constraints. In our computational study we consider the case in which two resources have disjoint shifts. This immediately triggers several instances infeasible if generalized precedence constraints are present that enforce jobs to be executed in parallel. Hence, we do not impose such constraints in the instances. Still, it is easy to extend our approach with such side constraints.

For such a turnaround, external workers are hired over the plannig horizon and must be paid for their availability—whether they are working or not. Hence, we want to level the resources, meaning that we need to compute the maximum capacity requirement of each resource and need to hire that amount of resource units at a minimum total resource cost. *Balancing* the resource usage, which means to flatten the ups and downs of the resource profile over time, is not an issue at our higher-level planning stage.

Problem definition

In Turnaround Scheduling we are given a set $\mathcal{J}$ of n jobs and a set $\mathcal{R}$ of renewable resources. Each job needs a finite number of resource units of exactly one resource $k \in \mathcal{R}$ for its processing.

Jobs may be executed in different modes $\mathcal{M}_j = \{1, \ldots, \bar{m}_j\}$, where $\bar{m}_j$ is the number of different modes of one job j. These modes (processing alternatives) of job $j \in \mathcal{J}$ are characterized by the number r_j of allocated resources and its resulting processing time p_j. We assume that $r_j \in \{r_{j1}, \ldots, r_{j\bar{m}_j}\}$ and $p_j \in \{p_{j1}, \ldots, p_{j\bar{m}_j}\}$ are integer and positive in each mode. Let $\mathcal{J}_k \subseteq \mathcal{J}$ denote the set of jobs that require resource $k \in \mathcal{R}$. Since each job requires exactly one resource, we can partition the set of jobs $\mathcal{J}$ into disjoint subsets $\mathcal{J}_1, \mathcal{J}_2, \ldots, \mathcal{J}_{|\mathcal{R}|}$.

The amount of work for processing a job $j \in \mathcal{J}$ in mode $m_j \in \mathcal{M}_j$ is given by $w_{jm_j} = r_{jm_j} \cdot p_{jm_j}$. We assume that the processing time is non-increasing and the amount of work is non-decreasing in the number of resource units. Hence, the *monotonicity properties*

$$p_{j\ell} \geq p_{jm} \qquad \text{and} \qquad w_{j\ell} \leq w_{jm}$$

hold for any mode $\ell, m \in \mathcal{M}_j$ with $\ell \leq m$.

Furthermore, each job $j \in \mathcal{J}$ is associated with an earliest start $\text{est}_j \in \mathbb{N}_0$ and a deadline $\text{lct}_j \in \mathbb{N}_0$ (latest completion time) which define the time-window $[\text{est}_j, \text{lct}_j)$ in which j must be processed. Precedence constraints are given by a directed acyclic graph $G = (\mathcal{J}, A)$ where the vertices correspond to jobs and where an edge $(i, j) \in A$ is given if job i precedes job j, i.e., $S_i + p_i \leq S_j$ must hold.

Each resource $k \in \mathcal{R}$ has an individual *calendar* $\mathcal{I}_k$ of working shifts $\mathsf{I} \subseteq \mathbb{R}$, which represents the *availability periods* of k. Hence, $\mathcal{I}_k = \{\mathsf{I}_1, \mathsf{I}_2, \ldots\}$. Each $\mathsf{I} \in \mathcal{I}_k$ can be written as $\mathsf{I} = [a, b)$, with $a < b$ integer. Jobs can only be processed non-preemptively during one availability period.

Definition 5.1. *A* schedule $(\boldsymbol{S}, \boldsymbol{m})$ *for the Turnaround Scheduling problem is given by a vector* $\boldsymbol{S} = (S_1, \ldots, S_n)$ *of integer start times* S_j *for each job* j *and by a vector of modes* $\boldsymbol{m} = (m_1, \ldots, m_n)$ *that assigns each job* j *a mode* $m_j \in \mathcal{M}_j$.

Definition 5.2. *A schedule* $(\boldsymbol{S}, \boldsymbol{m})$ *for the Turnaround Scheduling problem is* time-feasible *if it respects the release dates, deadlines and precedence constraints, i.e.,*

(i) $S_i + p_{im_i} \leq S_j$ *for all* $(i, j) \in E$, *and*

(ii) $\text{est}_j \leq S_j \leq S_j + p_{jm_j} \leq \text{lct}_j$ *for all* $j \in \mathcal{J}$.

The maximum completion time of a schedule $(\boldsymbol{S}, \boldsymbol{m})$, the *makespan*, is denoted by

$$C_{\max}(\boldsymbol{S}, \boldsymbol{m}) := \max_{j \in \mathcal{J}} \left\{ S_j + p_{jm_j} \right\}.$$

For a schedule $(\boldsymbol{S}, \boldsymbol{m})$, we denote by

$$r_k(\boldsymbol{S}, \boldsymbol{m}, t) := \sum_{j \in \mathcal{J}_k : S_j \leq t < S_j + p_{jm_j}} r_{jm_j}$$

the *resource utilization* of resource $k \in \mathcal{R}$ at time t, and by

$$R_k := \max_t r_k(\boldsymbol{S}, \boldsymbol{m}, t), \qquad \text{for all } k \in \mathcal{R}$$

the *maximum resource utilization* of resource $k \in \mathcal{R}$, i.e., the maximum number of resource units of resource k used at any point in time during the turnaround. The maximum resource utilization of a resource k may be bounded by a constant $\bar{R}_k \in \mathbb{N}$, which we call the *capacity* of that resource k. See Figure 5.1 for a schedule with availability periods and two resources.

Definition 5.3. *We call a schedule* $(\boldsymbol{S}, \boldsymbol{m})$ resource-feasible *iff*

(i) $R_k \leq \bar{R}_k$ *for all* $k \in \mathcal{R}$, *and*

(ii) for all $k \in \mathcal{R}$ *and* $j \in \mathcal{J}_k$ *with* $r_{jm_j} > 0$: $\exists\, \mathsf{I} \in \mathcal{I}_k : [S_j, S_j + p_{jm_j}) \subseteq \mathsf{I}$.

For each $k \in \mathcal{R}$, we are given a cost rate c_k that represents the cost per unit of resource k per time unit. The set of resources is partitioned into two disjoint subsets $\mathcal{R}^\ell$ and $\mathcal{R}^f$ depending on their payment type. Resources of type $k \in \mathcal{R}^\ell$ have to be paid during the entire turnaround period for the maximum amount needed. These are mainly external workers that are hired for the complete turnaround period and they must be paid for even if they are temporarily idle. Clearly, the goal of a project manager is to minimize the amount of paid idle time. In other words, the maximum resource utilization of that kind of resources should be minimized. We say that these resources *need to be leveled.* In contrast, resources of type $\mathcal{R}^f$ are paid for the actual work they perform. We say they are *free of leveling.* In our application, these are mainly internal resources. The job sets corresponding to $\mathcal{R}^\ell$ and $\mathcal{R}^f$ are denoted by $\mathcal{J}^\ell$ and $\mathcal{J}^f$, respectively, while all jobs needed by resource $k \in \mathcal{R}$ are denoted by $\mathcal{J}_k$.

Now, we can express the total cost of a schedule $(\boldsymbol{S}, \boldsymbol{m})$ as

$$\sum_{k \in \mathcal{R}^\ell} c_k \cdot R_k + \sum_{k \in \mathcal{R}^f} \sum_{j \in \mathcal{J}_k} c_k \cdot w_{jm_j}.$$

The first term is called the *resource availability cost* and represents the cost of resources that must be leveled, while the second term, called *resource utilization cost*, represents the cost of jobs that are free of leveling.

The task in Turnaround Scheduling is to find a schedule which is *time*- and *resource-feasible* and has minimum total cost. In practice, the first term clearly dominates the cost function. If we neglect the cost for jobs that are free of leveling, we speak of the *resource leveling problem.* This is the problem on which we focus.

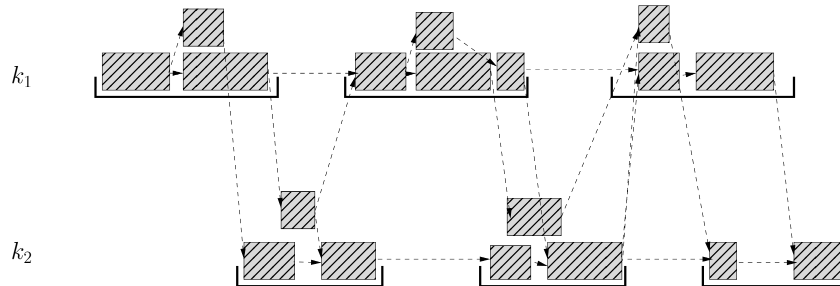


Figure 5.1: Schematic representation of a solution to a Turnaround Scheduling problem with two resources and three availability periods each.

Related work

The Turnaround Scheduling problem has many relationships with well established areas of scheduling. Following the extended $\alpha|\beta|\gamma$-classification scheme [36], we consider $MPS|prec, shifts|\sum c_k \cdot \max r_k(S, M, t)$ for multi-mode project scheduling with m renewable resources of unbounded capacity, with precedence constraints and working shifts, with the objective to minimize the total *resource availability cost*, i.e., minimizing $\sum_{k\in\mathcal{R}} c_k \cdot R_k$. We remark that in an MPS-environment usually multiple resources per job are needed. In our case, each job requires exactly one resource.

Multi-mode jobs are a key feature of Turnaround Scheduling problems. Such problems of the form $MPS|prec|C_{\max}$ have been investigated with renewable and non-renewable resources, with limited capacity, and makespan minimization, known as *multi-mode RCPSP*, see e.g., [85, 134]. In the notation scheme of Herroelen et al. [140], the Turnaround Scheduling problem is denoted by $m, 1|cpm, mu|rac$ for renewable resource with capacity m, common precedence relations and multi-mode jobs with the objective to minimize the resource availability cost.

Time-Cost Tradeoff Given a project network of jobs and precedence constraints, a job may be executed in different modes, each associated with a certain processing time and resource demand. The *time-cost tradeoff problem* asks for the relation between the duration of a project and its cost, while the cost are determined by the amount of non-renewable resource units which are necessary to achieve the project duration. For a nice survey we refer to De et al. [76]. Fixing either the project duration or the cost leads to the closely related optimization problems with the objective to minimize the other parameter; these problems are referred to as the *deadline problem* and the *budget problem*, respectively.

If the resource costs for the jobs are continuous, linear and non-increasing functions of the job processing times, then the deadline and the budget problem can be solved optimally in polynomial time as has been shown independently by Fulkerson [114] and Kelley [156]. Later, Phillips and Dessouky [208] give an improved version of the original algorithms in which iterative cut computations in a graph of critical jobs yield the piecewise linear time-cost tradeoff curve. This curve describes the tradeoff between project duration t and associated cost for all t. The running time is polynomial in the number of breakpoints of the optimal time-cost tradeoff curve, which may, however be exponential in the input size, see Skutella [238]. A breakpoint of such a piecewise linear function is a point in which the function is continuous but not differentiable. Elmaghraby and Kamburowski [106] generalized previous algorithms to solve a more general problem variant in which jobs may have release dates and deadlines and arbitrary time lags between them. They provide a combinatorial algorithm that iteratively computes minimum cost flows in a transformed network modeling the time-lags. Also other cost functions have been considered such as convex [4, 154, 169, 236] and concave functions [109].

In practical applications, the discrete version of this problem plays an important role. Here the processing time of a job is a discrete non-increasing function of the amount of the renewable resource allocated to it. This problem is known to be $\mathcal{NP}$-hard, see [77]. Skutella [237] derives approximation algorithms for the deadline and budget problem as well as bicriteria approximations, while Deineko and Woeginger [79] give lower bounds on the approximability of the problems.

Various exact algorithms and meta-heuristics have been implemented for the discrete

time-cost tradeoff problem. For an overview we refer to the book of Demeulemeester and Herroelen [85], chapter 8.

Time-Cost Tradeoff with capacity constraints Motivated by restrictions on resource capacities in real-world applications, the time-cost tradeoff problem has been investigated in problem variants with renewable as well as non-renewable resources of limited capacity. Such problems are also known as *multi-mode (resource constrained) project scheduling problems*; see e.g. [85].

Various versions of linear and discrete time-cost tradeoff related problems have also been considered in the theory of machine scheduling. Besides the available machines, there is an additional resource that allows to accelerate the processing of jobs. Approximation algorithms and even polynomial time approximation schemes have been derived. Among such problems stands scheduling with *controllable processing times* which concerns the allocation of non-renewable resource units, see the recent survey of Shabtay and Steiner [235]. Another direction is given by scheduling jobs with *resource dependent processing times* which assumes the presence of one discrete renewable resource, see Grigoriev et al. [128] and references therein. Given a fixed number of machines and the objective to minimize the makespan, scheduling *malleable jobs* where the duration of a job is determined by the number of machines allocated to it has been studied by e.g., [98, 151, 152, 173].

Resources with calendars and working shifts In real world applications, resources are rarely continuously available for processing. Working shifts, machine maintenance or other constraints may prohibit the processing in certain time intervals. Also in machine scheduling, various problems with limited machine availability have been considered. Regarding complexity and approximation results we refer to the survey by Schmidt [222].

In project scheduling, such constraints are known as break or shift calendars. Zhan [269] provides an exact pseudo-polynomial algorithm for computing earliest and latest start times in a generalized activity network that may contain minimum and maximum time lags, but no capacity bounds on the resources. His modified label-correcting algorithm respects jobs that may be preempted and those that must not. This algorithm has been modified into a polynomial time algorithm by Franck et al. [112]. In the same paper, priority-rule based heuristics are also provided for solving resource-constrained project scheduling problems where each job may require different resources.

Yang and Chen [266] consider a job-based and more flexible version of calendars represented by *time-switch constraints*. These constraints specify for any job several time windows in which it may be processed. They also extend the classical critical path method in order to analyze project networks when resource capacities are unbounded. Time-switch constraints have also been incorporated by Vanhoucke et al. [260] in the deadline version of the discrete time-cost tradeoff problem in order to model different working shifts. They present a branch-and-bound algorithm which has later been improved by Vanhoucke [255]. Experimental results are shown for instances with up to 30 jobs and up to seven different processing modes. Vanhoucke and Debels [258] investigate a further extension of the deadline problem with time-switch and other side constraints with the objective to maximize the net present value.

Resource Leveling Typical goals in project management are the minimization of the total project duration (makespan), the maximization of net present value or more service oriented goals such as minimizing waiting time or lateness. In certain applications the objective functions are based on resource utilization; see e.g., Neumann et al. [193]. In particular, when resource units are rented for a fixed time period, they should be utilized evenly over this time.

The resource leveling problem with single-modes per job, which is denoted by $PS|temp|\sum c_k \max r_k(S,t)$ with general temporal constraints. It has been considered earlier under the name *resource investment problem.* The special case without generalized precedence constraints $PS|prec|\sum c_k \max r_k(S,t)$ has been considered e.g., by Demeulemeester [82] and Möhring [187]. These authors competed on the same instance set which contained about 16 jobs and four resources, with a time horizon between 47 and 70. Further computational studies were done containing 15 to 20 jobs and four resources. In the same setting, Drexl and Kimms [96] propose lower bound computations, one based on Lagrangian relaxation and another based on a column generation procedure, where variables represent schedules as in our approach. Instances consisting of 20 jobs with low processing times can be handled; for 30 jobs the Lagrangian relaxation wins against the column generation approach.

Harris [132] develops a critical path based heuristic for resource leveling of precedence constrained jobs with fixed processing times and no side constraints. Neumann [194] presents a heuristic and exact algorithms for the resource leveling problem with temporal and resource constraints. Gather et al. [118] describe an enumerative tree search for the resource leveling problem in which the total squared utilization cost need to be minimized. In a number of earlier publications such as [15, 35, 102, 207], heuristics and exact algorithms for simplified problem versions can be found.

Benchmark instances For a computational benchmarking of project scheduling problems, different problem sets are available in the PSPLib [162]. There, several variants of RCPSP and of resource investment problems can be found. For the RCPSP single-mode case, test sets containing 60 jobs could not be solved to proven optimality by various researchers. In the multi-mode case, instances with 30 jobs are not solved yet. For the resource investment problem, test sets containing 10, 20, or 30 jobs are available, but they do not contain working shifts, are in single-mode or include time-lags.

Even though none of these problem variants is immediately suited for a direct comparison, they are similar to ours, and the mentioned instances inspired us when generating our own test set as described in Section 5.4.

Complexity results The resource leveling problem contains all features of the basic RCPSP and is therefore strongly $\mathcal{NP}$-hard and challenging when looking for good solutions. From a theoretical point of view, simplified versions and their approximability have been studied. Günther et al. [131] present approximability results for malleable tasks with precedence constraints. Cieliebak et al. [62] consider a variant of interval scheduling in which they aim for minimizing the maximum number of used resource units. They derive approximation algorithms and hardness results. Hence, resource leveling problems in particular for large scale instances need further research to be solved more efficiently. Our approach of splitting the planning horizon into several sub-schedules may also lead to better results for related problems.

5.2 Turnaround heuristics

Heuristics for large-scale scheduling problems are highly demanded in practice. We present a parallel and a serial schedule generation scheme adapted to the multi-mode case. Both heuristics are also used as subroutines in a resource leveling heuristic.

The heuristics presented next will be used in our study as primal heuristics during branch-and-bound search.

Parallel SGS

A *parallel SGS* first schedules a set of jobs which are available at time zero as early as possible. Then, the next possible start time is considered with a new set of jobs that which available. Jobs can only be scheduled if their predecessors are completed. Unscheduled jobs whose predecessors have been scheduled are called *ready jobs*.

For each resource $k \in \mathcal{R}^\ell$ we maintain a vector of jobs $J_k = (j_{k_1}, .., j_{k_{|J_k|}})$ that use this resource and have no unscheduled predecessors, together with a lower bound t_k on the next feasible start $\text{est}'_j \geq \text{est}_j$ of any job $j \in J_k$. If J_k is empty, we set t_k to infinity.

The heuristic loops over t, which increases to the minimal t_k in each iteration. A subset $J \subset J_k$ of constant size s (we chose $s = 6$ in our computational studies) is scheduled so that the overall completion time is kept small (line 9). This is accomplished by trying all mode combinations for J recursively, and bounding recursion using the currently shortest feasible solution found in this way. If s is small, this can be done quickly. Finally, for each scheduled job in J, those successors which become ready, are added to the corresponding set J_k, each t_k is updated, and the next iteration begins. This process continues until all jobs are scheduled, or a makespan violation occurs.

Algorithm 10: Parallel SGS for multi-mode jobs

Input: Set of jobs $\mathcal{J}$ to be scheduled and max. total duration T.
Output: A schedule $(\boldsymbol{S}, \boldsymbol{m})$, or that no solution is found.

```
1  foreach  k ∈ R do
2  |   Let J_k ⊂ J be the set of jobs that use resource k, and are ready.
3  |   t_k := min_{j∈J_k} est'_j.
4  end
5  t := 0
6  while  t < T and not all jobs are scheduled do
7  |   ℓ := argmin_k t_k, and t := t_ℓ.
8  |   m := min(s, |J_ℓ|) with s a small constant and J := {j_ℓ1, .., j_ℓm}.
9  |   Schedule jobs in J such that max_{j∈J} C_j is minimal.
10 |   Add all successors of jobs to J that become ready to their respective J_k.
11 |   J_ℓ := J_ℓ \ J.
12 |   Update t_k for all changed J_k as in Step 3.
13 end
```

Serial SGS

In a serial SGS, jobs are scheduled according to a topological ordering. Therefore, this is also called a list schedule. The ordering can be done with respect to the bounds of the variables or with respect to LP solution values. According to that ordering, an

earliest start schedule is computed respecting resource capacities by shifting jobs to the earliest feasible start time where enough capacity is available. We use the values of the LP solution as input and obtain a fast primal heuristic that can be used throughout branch-and-bound search.

Our heuristic sets the maximal capacity of each resource to the LP solution value rounded up, and fix the earliest and latest start and completion times for each job to the local bounds of the corresponding variables. Then, we perform list scheduling with jobs sorted by earliest completion times in the LP solution, and each job's mode is chosen as the one matching $C_j^\star - S_j^\star$ best. Again, $S_j^\star, C_j^\star$ denote the start and completion times of job j in the LP solution.

If a feasible schedule is found new columns (one per shift) representing that schedule are added to the master problem, in order to reduce the total number of pricing steps.

Resource leveling heuristic

We now describe the resource leveling heuristic, and how the serial and parallel SGS tie into the framework of the leveling procedure. The leveling heuristic uses a binary search on the capacity bounds of the resources, greedily selecting the resource whose upper bound is to be improved in each iteration. This selection is based on a parameter μ_k, measuring how badly a resource k is leveled.

$$\mu_k = (R_k \cdot T)/(\sum_{j \in \mathcal{J}} r_j^{\min} \cdot p_j^{\max}).$$

One iteration of the binary search consists of trying to find a feasible schedule for the current lower (LB_k) and upper (UB_k) bounds of the resource capacity variables R_k. These bounds for the selected resource $k^\star$ are set to $(\mathrm{UB}_{k^\star} + \mathrm{LB}_{k^\star})/2$, $\mathrm{UB}_{k^\star}$ and $\mathrm{LB}_{k^\star}$ being the upper and lower bounds on the capacity of resource $k^\star$, while all other resource bounds remain fixed. We try serial SGS first, and on failure fall back to the parallel SGS to prove the bounds feasible. If neither serial nor parallel SGS yield a feasible schedule, we consider the current upper bound for the selected resource as a new lower bound, and the next iteration begins.

Algorithm 11: Resource Leveling

Input: Set $\mathcal{R}$ of resources to be leveled, project duration T.
Output: Leveled resource utilization R_k for each resource $k \in \mathcal{R}$.

```
1 Set LB_k and UB_k to initial values for each resource k ∈ R.
2 while ∃ k ∈ R : LB_k < UB_k do
3 |   Choose resource k* ∈ R with LB_k* < UB_k* and μ_k* maximum.
4 |   Perform binary search using serial and parallel SGS in order to decrease the
  |   capacity bounds of k*.
5 end
```

5.3 Mixed Integer Programming formulations

For solving large-scale scheduling problems, Mixed Integer Programming (MIP) is not considered as primary choice since the Linear Programming (LP) relaxations may be weak. Huge numbers of variables and constraints may result in high computation times

and memory failures for solving even only the LP relaxation. The approach we propose demonstrates that more sophisticated algorithmic techniques can be a partial remedy for these issues. We start by presenting a plain MIP in Section 5.3.1 that will be solved by `CPLEX`. Afterwards the basics of column generation are introduced as needed here in Section 5.3.2 including a MIP model of exponential size where variables correspond to subschedules per working shift.

5.3.1 Obstacles of MIP formulations

In the following we present a mixed integer programming formulation of the Turnaround Scheduling problem for a given project deadline T. It borrows from the classical time-indexed formulation based on start times for RCPSP as introduced in Section 1.3.1 and incorporates the multi-mode characteristics of jobs. We use binary decision variables x_{jmt} that indicate whether job $j \in \mathcal{J}$ starts in mode $m \in \mathcal{M}_j$ at time $t \in \{0, 1, \dots, T-1\}$.

We model resource calendars implicitly using start-time dependent processing times. In a preprocessing step we compute for each job $j \in \mathcal{J}$ the processing time p_{jmt} it has when starting at time t in mode $m \in \mathcal{M}_j$. If a job cannot be scheduled with respect to calendars at time t, we set the corresponding variable x_{jmt} to zero. The resource demands of job j in mode m for resource k is denoted by r_{jmk}.

$$\min \sum_k c_k \cdot R_k \tag{5.1}$$

$$\text{subject to} \quad \sum_t \sum_{m \in \mathcal{M}_j} x_{jmt} = 1 \qquad \forall\, j \in \mathcal{J} \tag{5.2}$$

$$\sum_t \sum_{m \in \mathcal{M}_j} t \cdot x_{jmt} = S_j \qquad \forall\, j \in \mathcal{J} \tag{5.3}$$

$$\sum_t \sum_{m \in \mathcal{M}_i} (t + p_{im}) x_{imt} \leq \sum_t \sum_{m \in \mathcal{M}_j} t x_{jmt} \qquad \forall\, (i,j) \in E \tag{5.4}$$

$$\sum_{j \in \mathcal{J}} \sum_{m \in \mathcal{M}_j} \sum_{\substack{\tau = t - p_{jm} + 1 \\ \tau \geq 0}}^{t} r_{jmk} \cdot x_{jm\tau} \leq R_k \qquad \forall\, k, t \tag{5.5}$$

$$S_j \geq 0, \forall\, j \in \mathcal{J} \qquad R_k \geq 0, \forall\, k \in \mathcal{R} \tag{5.6}$$

$$x_{jmt} \in \{0,1\} \qquad \forall\, t, j, m \in \mathcal{M}_j \tag{5.7}$$

Constraint (5.2) assures that each job starts exactly once in one of its modes. The (continuous) start time S_j of job $j \in \mathcal{J}$ is linked to the binary variables x_{jmt} in equations (5.3) and is therefore implicitly integral. Hence, we do not impose any integrality constraints on start time variables. Because of (5.4) every pair of jobs $(i, j) \in E$ respect the precedence constraints. Finally, inequalities (5.5) guarantee that the resource constraints are met. We decide about the (continuous) resource capacities $R_k \geq 0$ for each resource k, such that the total resource availability cost is minimized via the objective function (5.1).

Due to the time expansion, time-indexed formulations for scheduling problems are usually hopeless for large problem instances. However, small instances can be solved using integer programming solvers such as CPLEX, and we can thus evaluate the performance of our algorithm by comparing the computed solution with an optimal solution for such instances.

Depending on several factors, such as network complexity (the density of G) or the time discretization considered, this formulation may give poor lower bounds. Very often, we experience that in an optimal solution to the LP relaxation only few binary variables are fractional, but the points in time used in the convex combination (5.3) to yield the actual start time S_j of a job are far apart from one another. We will informally call this a “smearing” of start time variables. This smearing gives us irrelevant information about the schedule and we lose all structure in the model.

Furthermore, branching on the binary variables leads to an unbalanced search tree, since branching to zero is a very weak decision if the job can be scheduled one time unit earlier or later (that is, the decision essentially has no effect). Thus, one needs a more sophisticated branching rule that is aware of the linking of continuous variables S_j and binary variables x_{jmt} and that prefers branching on the start time variables S_j. Therefore, natural branching candidates are start time variables whose corresponding binary variables are fractional. It turns out that this intuition goes very well with our approach.

5.3.2 Exponential formulation

Column Generation procedures come with a huge amount of variables (exponentially many), but this disadvantage can sometimes be beaten by tighter dual bounds. On the other hand, the pricing problems become harder to solve, such that IP and CP techniques are needed for them. Before stating the new model, we start by introducing the concept of Dantzig-Wolfe decomposition as needed here.

5.3.2.1 Dantzig-Wolfe decomposition

A Dantzig-Wolfe decomposition makes use of the problem structure and aggregates a bunch of variables into maybe an exponential number of new master variables that capture all possible combinations of the original variable values. To overcome the huge amount of variables, these new variables need to be generated and added to the LP relaxation.

Starting from a standard MIP formulation with appropriate sized matrices B, D and vectors b, c, d and variable vector x, the Dantzig-Wolfe decomposition works as follows: We apply Minkowski-Weyl theorem and express a polytope $P = \{x \in \mathbb{Z}^n \mid Bx \leq b\}$ as convex combination of its extreme points $p_i \in \mathbb{Z}^n$. This is possible for polytopes but not for polyhedrons. Hence, we substitute:

$$x := \sum_i p_i \lambda_i, \qquad \sum_i \lambda_i = 1, \qquad \lambda_i \geq 0.$$

An IP before and after applying Dantzig-Wolfe decomposition is given in Figure 5.2.

Such a decomposition works well if the matrix B imposes nice structural properties, like block structures or being totally unimodular. Matrix D can be interpreted as those constraints that link the variables between blocks with each other. We show that this approach is useful for our scheduling problem even though the pricing problems involved are complex and strongly NP-hard on their own. In our model we identify the blocks of matrix B as the resource constraints per shift and matrix D, the linking constraints, corresponds to the precedence constraints (5.4) and constraints (5.2) that ensure that each job is executed exactly once.

Original formulation:

$$\begin{aligned} \min \quad & c^t x \\ \text{subject to} \quad & Bx \leq b \\ & Dx \leq d \\ & x \in \mathbb{Z}^n \end{aligned}$$

DW-reformulation:

$$\begin{aligned} \min \quad & \sum_i (c^t p_i) \cdot \lambda_i \\ \text{subject to} \quad & D \sum_i p_i \lambda_i \leq d \\ & \sum_i \lambda_i = 1 \\ & \lambda \geq 0 \end{aligned}$$

Figure 5.2: Dantzig-Wolfe decomposition for a general MIP by substituting $x := \sum_i p_i \lambda_i, \sum_i \lambda_i = 1, \lambda_i \geq 0, \forall\, i$, where $p_i \in \{x \in \mathbb{Z}^n \mid Bx \leq b\}$.

5.3.2.2 Master problem: A model based on shift configurations

In order to reduce the effects of "losing the timing information" because of the smearing of variables, we propose a model which exploits the problem structure by decomposing the time horizon into the availability periods of resources. Based on the calendar for each resource, every working shift represents a smaller subproblem for which sub-schedules, or *configurations*, are generated independently for each resource. These configurations are linked by global constraints ensuring that exactly one configuration is chosen for each working shift.

Definition 5.4. *A configuration ξ for an interval $I \in \mathcal{I}_k$ of resource k is a schedule for a subset $J \subseteq \mathcal{J}$ that require resource k. Each such job $j \in J$ is specified by its start time $S_{j\xi}$ and its completion time $C_{j\xi}$. The maximum number of resources simultaneously used by these jobs is denoted by $\bar{r}_\xi$.*

For each configuration ξ for a particular shift $I \in \mathcal{I}_k$ we introduce a binary variable x_ξ which indicates whether configuration ξ is chosen or not. We use the short hand notation $j \in \xi$ to express that job j is executed in the shift corresponding to configuration ξ. Note that the mode of each job is determined by the respective start and completion times. The so-called *master problem* reads:

$$\min \sum_k c_k \cdot R_k \tag{5.8}$$

$$\text{subject to} \quad C_i \leq S_j \qquad \forall\, (i,j) \in E \tag{5.9}$$

$$S_j = \sum_{\xi : j \in \xi} S_{j\xi} \cdot x_\xi \qquad \forall\, j \in \mathcal{J} \tag{5.10}$$

$$C_j = \sum_{\xi : j \in \xi} C_{j\xi} \cdot x_\xi \qquad \forall\, j \in \mathcal{J} \tag{5.11}$$

$$\sum_{\xi : \xi \in I} \bar{r}_\xi \cdot x_\xi \leq R_k \qquad \forall\, k\, \forall\, I \in \mathcal{I}_k \tag{5.12}$$

$$\sum_{\xi : j \in \xi} x_\xi = 1 \qquad \forall\, j \in \mathcal{J} \tag{5.13}$$

$$R_k, S_j, C_j \geq 0 \qquad \forall\, k, j \in \mathcal{J} \tag{5.14}$$

$$x_\xi \in \{0,1\} \qquad \forall\, \xi \tag{5.15}$$

Each job is executed in exactly one configuration by (5.13). The start and completion times for each job are computed from the chosen configurations via the linking constraints (5.10) and (5.11). Constraint (5.13) ensures, that for each job exactly one mode (from one configuration) can be chosen in an integer optimal solution. Hence, only feasible modes are enumerated. Constraints (5.9) model the precedence relations between jobs. These could be directly expressed by substituting S_j and C_j from the linking constraints, but (5.10) and (5.11) are helpful in the pricing problem (Section 5.3.2.3) where they penalize or encourage certain start or completion times of jobs. Constraints (5.12) link resource consumptions to the capacities and ensure that the total number of resources available over the planning horizon is at least the maximum number of resources needed in each shift.

5.3.2.3 Column generation: Pricing problem

Since the number of feasible configurations is exponential in the number of jobs, we solve the LP relaxation by column generation. That is, we start with a very small (e.g., heuristically generated) subset of configuration variables, and dynamically add more variables to the model until one can prove that no more promising variables exist. This optimality proof is given—as in the standard simplex method—via non-negativity of reduced costs of all configuration variables. We now describe the *pricing subproblem* which is used to generate configuration variables with negative reduced cost or to prove that none exists.

We must solve a pricing problem for every shift $I \in \mathcal{I}_k$ of each resource k. We denote the dual variables of constraints (5.10), (5.11), (5.12), and (5.13) by s_j, c_j, ρ_{kI}, and π_j, respectively. We formulate a pricing problem for each shift I in which only the subset $J \subseteq \mathcal{J}$ of the jobs that can be scheduled in I needs to be considered. The objective function (5.16) reflects minimizing the reduced cost.

$$\max \quad \sum_j \pi_j \cdot X_j - \sum_j c_j \cdot C_j - \sum_j s_j \cdot S_j - \rho_{kI} \cdot \bar{r}_k \tag{5.16}$$

$$\text{subject to} \quad \sum_{t \in I} \sum_{m \in \mathcal{M}_j} x_{jmt} = X_j \qquad \forall\, j \in J \tag{5.17}$$

$$\sum_{t \in I} \sum_{m \in \mathcal{M}_j} t \cdot x_{jmt} = S_j \qquad \forall\, j \in J \tag{5.18}$$

$$\sum_{t \in I} \sum_{m \in \mathcal{M}_j} (t + p_{jm}) \cdot x_{jmt} = C_j \qquad \forall\, j \in J \tag{5.19}$$

$$\sum_{j \in J} \sum_{m \in \mathcal{M}_j} \sum_{\substack{\tau = t - p_{jm} + 1 \\ \tau \in I}}^{t} r_{jmk} \cdot x_{jm\tau} \leq \bar{r}_k \qquad \forall\, t \in I \tag{5.20}$$

$$\bar{r}_k \geq 0 \tag{5.21}$$

$$x_{jmt} \in \{0, 1\} \qquad \forall\, j \in J, m \in \mathcal{M}_j, t \in I \tag{5.22}$$

$$X_j \in \{0, 1\} \qquad \forall\, j \in J \tag{5.23}$$

This is again a scheduling problem with a non-regular objective function where a new configuration ξ for a specific shift $I \in \mathcal{I}_k$ of resource k is generated. It must be decided,

see constraint (5.17), whether a job j corresponding to the binary decision variable X_j is executed in this shift or not, and if so, which mode $m \in \mathcal{M}_j$ is used. Constraints (5.18) and (5.19) encode the start and completion times of jobs according to the chosen mode assignment. Resource capacity constraints (5.20) have to be satisfied as before and the number of resources needed in this shift is again a decision variable, denoted by $\bar{r}_k$. The objective function value is increased by π_j if a job is taken into the configuration and by multiples of s_j and c_j according to the start and completion times. With each unit increase of resource capacity the objective value decreases by a factor of ρ_{kI}.

The pricing problem (5.16)–(5.23) is $\mathcal{NP}$-hard as it contains a resource leveling problem as special case. This can be seen as follows: Set all π_j to a value large enough to ensure that each job must be scheduled, and let s_j and c_j be zero for all j. Then, we need to schedule all jobs in that interval at a minimum resource cost $\rho_{kI} \cdot \bar{r}_k$. This negative complexity result justifies solving the pricing problem as a mixed integer program itself as no better algorithm is known.

Note that so far only the LP relaxation of the master problem (5.8)–(5.15) is solved to optimality, and integrality of the configuration variables x_ξ still needs to be ensured by branching decisions, see Section 5.3.3.1. This leads us to a *branch-and-price* algorithm, i.e., solving the LP relaxation in each node of the branch-and-bound tree by column generation.

5.3.3 Branch-price-and-cut algorithm

A solution to the original problem is given by the resource capacities R_k, and an assignment of start times S_j and completion times C_j for each job j. The mode is given by the unique mode m_j, such that $p_{jm_j} = C_j - S_j$. In a modern branch-and-price context one tries to branch on these *original variables*, instead of on the master variables [87]. A main reason is that branching decisions on master variables are hard to respect in the pricing problems, i.e., forbidden configurations must not be re-generated. Moreover, branching decisions on master variables imply additional constraints that give rise to additional dual variables that need to be respected in the pricing problem. On the other hand, branching constraints formulated on the original variables only affect the subproblems' objective function, not their structure, which is very desirable [87, 88].

5.3.3.1 Branching scheme

Preliminary experiments revealed that not all branching decisions are of equal importance, so we choose to branch on the most important variables first. Only when these are already integer, we branch on the second-important class, and so forth. The order we choose is R_k, S_j, and then C_j. That is, branching on fractional resource capacity values has the largest impact. Start and completion time variables are considered as branching candidates only if any corresponding binary configuration variable is fractional. After the resource capacities are fixed in the search tree, a start time variable S_j with LP solution value $S_j^\star$ is selected. The node is split into two child nodes with $S_j \leq \lfloor S_j^\star \rfloor$, and $S_j \geq \lceil S_j^\star \rceil$, respectively. Completion times are handled accordingly. This scheme is used together with some propagation rules to overcome the "smeared" LP solutions and to create a more balanced search tree.

5.3.3.2 Propagation

Domains of variables can be tightened due to logical implications given by the constraints, and/or already fixed variables. In a branch-and-price approach, domain propagation is also used to preserve the validity of the LP relaxation after branching, see [215], i.e., all master variables that are not consistent with the local bounds need to be set to zero. Furthermore, in the pricing problem only variable assignments that are locally feasible in the current node can be chosen and need to be preprocessed before setting up the pricing problem.

In the area of scheduling problems, a large variety of propagation algorithms is known that detect these infeasible start times and perform variable bound adjustments see Section 2.1.3.2. We extended edge-finding to the multi-mode case, by using the minimum energy of all modes for each job, which naturally seems to give weaker bounds. This is balanced by the fact that jobs are not preemptive, may not cross shift-bounds and obey precedence constraints which enables further propagation of start and completion times. We use this algorithm in every node of the branch-and-bound tree, prior to calling the pricing problem and cutting plane separation (see below).

5.3.3.3 Cutting planes

State-of-the-art MIP solvers heavily rely on additional valid inequalities ("cutting planes") in order to improve the dual bound and by that prune unpromising nodes of the branch-and-bound tree. Adding cutting planes to the master problem is possible, but raises again (as with branching decisions) the issue of how to respect the additional dual variables in the pricing problem. It is technically easier to formulate valid inequalities on the *original* variables and add their Dantzig-Wolfe reformulation, i.e., in our case their translation to configuration variables, to the master problem. Again, this only changes the objective function of the pricing problem, see again [88] for details. Incidentally, this is a good situation for us as the literature knows several cutting planes for various scheduling problems, all of them formulated on variables with a meaning as in the standard MIP (5.1)–(5.7). Here, we generalize the precedence inequalities as introduced in Section 4.2 where jobs have a fixed processing time:

$$\forall\,(i,j) \in E, \tau: \qquad \sum_{t \geq \tau} x_{it} + \sum_{t < \tau + p_i} x_{jt} \leq 1. \tag{5.24}$$

These cuts read as follows: For any point in time τ, in an end-to-start precedence relation between jobs i and j, either job i starts the latest in τ or job j starts before $\tau + p_i$. As multiple modes are present we compute the minimum processing time $p_i^{\min}$ a job i may have locally, i.e., the minimum over all of its modes. Building on these cuts, we obtain the following cuts in our master problem:

$$\forall\,(i,j) \in E, \tau: \qquad \sum_{\xi : S_{i\xi} \geq \tau} x_\xi + \sum_{\xi : S_{j\xi} \leq \tau + p_i^{\min}} x_\xi \leq 1. \tag{5.25}$$

As desired, the nature of the pricing problem (i.e., its constraints) does not change, only the coefficients of the objective function need to be updated. For each constraint $(\tau, (i,j))$ that is added to the relaxation, we introduce the dual variable $\mu_{\tau ij}$ and add it to the objective function of the pricing problem. If $S_{i\xi} \geq \tau$ or if $S_j < \tau + p_i^{\min}$, the cost coefficient of variable x_{imt} becomes:

$$\sum_{(i,j)\in E}\sum_{\tau\geq t}\mu_{\tau ij} + \sum_{(k,i)\in E}\sum_{\tau<t+p_{im}}\mu_{\tau ki} \tag{5.26}$$

The cutting planes are added to the master problem after new configuration variables have been generated and each configuration variable is introduced into the corresponding constraints. Overall, this gives a branch-price-and-cut algorithm.

These cuts can be extended to generalized precedence constraints, as we show next. Generalized precedence constraints are of the form start-to-start (GP_1), end-to-start (GP_2), start-to-end (GP_3) and end-to-end (GP_4). Let $\delta_{ij} \in \mathbb{Z}$, then we can include these generalized precedence relations in our master problem by adding the following inequalities based on start and completion time variables (S_j, C_j), and also separate strengthened inequalities, similar to (5.24), based on the configuration variables (x_ξ).

$(i,j) \in GP_1 \Leftrightarrow S_i + \delta_{ij} \leq S_j$ or:

$$\sum_{\xi:S_{i\xi}\geq\tau} x_\xi + \sum_{\xi:S_{j\xi}<\tau+\delta_{ij}} x_\xi \leq 1 \qquad \forall\,\tau,$$

$(i,j) \in GP_2 \Leftrightarrow C_i + \delta_{ij} \leq S_j$ or:

$$\sum_{\xi:C_{i\xi}\geq\tau} x_\xi + \sum_{\xi:S_{j\xi}<\tau+\delta_{ij}} x_\xi \leq 1 \qquad \forall\,\tau,$$

$(i,j) \in GP_3 \Leftrightarrow S_i + \delta_{ij} \leq C_j$ or:

$$\sum_{\xi:S_{i\xi}\geq\tau} x_\xi + \sum_{\xi:C_{j\xi}<\tau+\delta_{ij}} x_\xi \leq 1 \qquad \forall\,\tau,$$

$(i,j) \in GP_4 \Leftrightarrow C_i + \delta_{ij} \leq C_j$ or:

$$\sum_{\xi:C_{i\xi}\geq\tau} x_\xi + \sum_{\xi:C_{j\xi}<\tau+\delta_{ij}} x_\xi \leq 1 \qquad \forall\,\tau.$$

Now, for each cut encoded by $(\tau,(i,j)) \in GP_\ell$ we need to add the value of the dual variable $\mu^\ell_{\tau ij}$ to the objective function for all types of generalized precedence relations GP_ℓ, $\ell = 1,\ldots,4$ according to the following rules. The cost coefficient of variable x_{imt} in the pricing problem is increased by:

GP_1:

$$\sum_{(i,j)\in GP_1}\sum_{\tau:t\geq\tau}\mu^1_{\tau ij} + \sum_{(k,i)\in GP_1}\sum_{\tau:t<\tau+\delta_{ki}}\mu^1_{\tau ki},$$

GP_2:

$$\sum_{(i,j)\in GP_2}\sum_{\tau:t+p_{im}\geq\tau}\mu^2_{\tau ij} + \sum_{(k,i)\in GP_2}\sum_{\tau:t<\tau+\delta_{ki}}\mu^2_{\tau ki},$$

GP_3:

$$\sum_{(i,j)\in GP_3}\sum_{\tau:t\geq\tau}\mu^3_{\tau ij} + \sum_{(k,i)\in GP_3}\sum_{\tau:t+p_{im}<\tau+\delta_{ki}}\mu^3_{\tau ki},$$

GP_4:

$$\sum_{(i,j)\in GP_4}\sum_{\tau:t+p_{im}\geq\tau}\mu^4_{\tau ij} + \sum_{(k,i)\in GP_4}\sum_{\tau:t+p_{im}<\tau+\delta_{ki}}\mu^4_{\tau ki}.$$

5.4 Computational study

In this section we compare the outcome of our branch-price-and-cut algorithm against a presolved time-indexed formulation solved by the MIP solver CPLEX. In particular, we are interested in the dual bounds that are obtained in the root node by either algorithm since these bounds indicate the strength of the polyhedral description. Furthermore, we give experimental evidence that the branching strategy and on some instances a careful use of precedence inequalities play an important role to efficiently solve a large number of the benchmark problems.

5.4.1 Benchmark instances

There is no publicly available test set of instances reflecting the setup of our problem. When compiling our own test set, the existing instances in the PSPLib [162] guided our design. Our set is composed of two sets of job scenarios, with 50 instances each. Each job can run in three different modes, using one to three units of its resource, with durations ranging from five to twelve. The first set, denoted by N50E70 contains 50 jobs and 70 precedence constraints, whereas the second set N50E100 contains 50 jobs and 100 precedence constraints. The maximal width W of the precedence graph is six, which is achieved by constructing W chains of length $|\mathcal{J}|/W$, and randomly choosing the remaining edges.

There are two different resources which come in five calendar configurations, called C1 to C5. These calendars are described schematically in Figure 5.3. In the top row, calendars C1 to C3 are shown. In each of these, the length of the shifts is 60. In C1 and C3 shift breaks are 60 units long, in C2 only 20. Both resources are available at the same time in C1, while in C3 availability periods are complementary. In C2 the second resource is offset at 40 units. Calendars C4 and C5 show shifts with length 20 and breaks having length five. In C5 one of the resources is offset by ten. All scenarios are tested with each of the five different calendars. Time horizons were chosen by computing a minimal and maximal makespan heuristically using an earliest start list scheduling policy, and averaging these. The first makespan comes by scheduling all jobs at their highest resource usage and the second run is carried out by assigning the fewest number of resources to each job.

Experimental setup

All experiments were done on Intel Core™ i7-870 PCs (2.93 GHz, 8MB cache, 8GB memory) running Linux 2.6.34 (single thread). Each test run had a time-limit of 30 minutes. Our C++ implementation is based on SCIP 2.0.1 [232] to perform the branch-and-price process, with custom plug-ins for our heuristics, branching rules, cutting plane

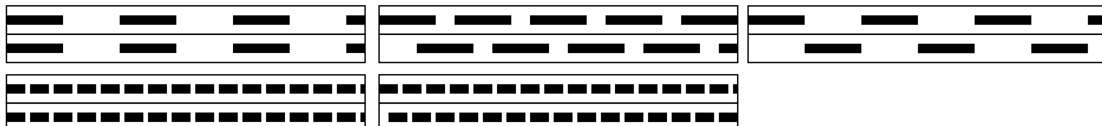


Figure 5.3: Calendar configurations C1–C3 (top) and C4 and C5 (bottom) used in our test set. Black bars symbolize the temporal location of shifts.

separation, and column generation. For the standard MIP (5.1)–(5.7) we used CPLEX 12.2 on the same machine, with default parameter settings, again single thread. Up to two threads were run in parallel on not entirely idle machines, so run time differences of 5% are probably "noise."

Usefulness of the heuristics Previous results showed that it is important for CPLEX and our branch-and-price framework to have a good initial solution, whereas the use of the heuristic throughout search is negligible, see [67].

Separation procedure Since there are $m \cdot T$ many precedence cuts (5.25), we need a good separation oracle. We pursue two approaches in order to add these cuts into the model. In our first approach we only check for a precedence pair (i, j) at time point $\tau_{greedy} = \lceil (C_i + S_i)/2 \rceil$ where equation (5.25) holds or is violated by more than some ϵ. In the second approach, we find the optimum point τ_{best} such that the left-hand side of equation (5.25) is maximally violated. This can be done by sorting the summands of equation (5.25) for each time point τ such that for each value τ the left-hand side can be computed in linear time (after sorting). This procedure is pseudopolynomial in the number of time points.

Settings We compare the outcome of a CPLEX run on MIP model (5.1)–(5.7) to different settings of our branch-and-price approach. Initially, we do not separate cuts, denoted by "nocuts." Then, we evaluate how to separate the cuts. First, we separate them already in the root node, and second as it will turn out to be profitable, after resource capacity variables are fixed. Since it may not always be beneficial to check for the largest violation, we use a promising guess as point in time: "τ_{greedy}" for each precedence pair (i, j). Searching for the highest violation is denoted by "τ_{best}" which is computed as described above. Furthermore, these cuts are only separated if a certain threshold is exceeded. We call this the tolerance "tol." and try tolerances 10^{-4}, 0.1, 0.3, 0.5 and 0.8, which range from a very small violation at which the cut is separated to a very high violation. In some tables 0.0 denotes the parameter setting of 10^{-4}.

5.4.2 Results

Very often a Dantzig-Wolfe decomposition can improve the lower bound obtained from the standard linear relaxation by far. Nevertheless, the cutting plane algorithms of commercial MIP solvers are sophisticated alternatives to improve the dual bounds. In Figure 5.4 we see that the average improvement per calendar of the decomposed LP relaxation (5.8)–(5.15) compared to the LP relaxation of (5.1)–(5.7) lies between 20% to 65% throughout all instances. The displayed average deviations and the minimum and maximum values of that improvement show the strength of our relaxation.

To evaluate the strength of generic cutting planes added in the root node of the commercial MIP solver, we compare the root dual bounds in Figure 5.5 after cutting planes have been added by CPLEX and our solver. Our improvement is no longer that dominating as in Figure 5.4 but still for several of the hard instances in calendar setting $C1$, there is a 28% improvement on the average of the root dual bound compared to CPLEX.

Now, we compare our branch-and-price framework to a standard MIP approach in terms of absolute number of solved instances and afterwards, we discuss the effect of

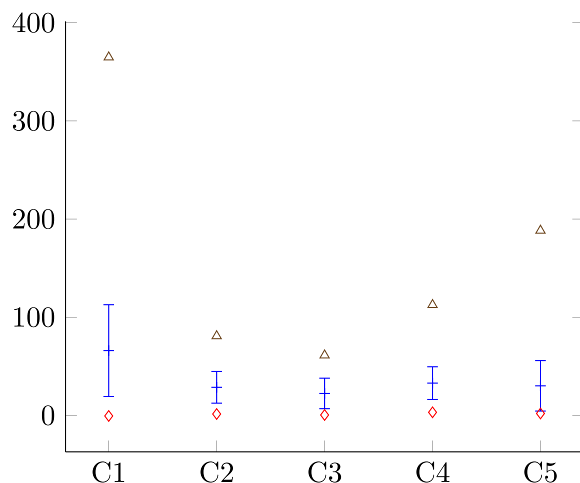


Figure 5.4: Percentage of improvement of the lower bound obtained from our branch-and-price formulation (5.8)–(5.15) over the standard LP relaxation of (5.1)–(5.7) as computed by CPLEX. The minimum (diamond), maximum (triangle), mean, and standard deviation of the improvement in percent of setting "no cuts" is shown.

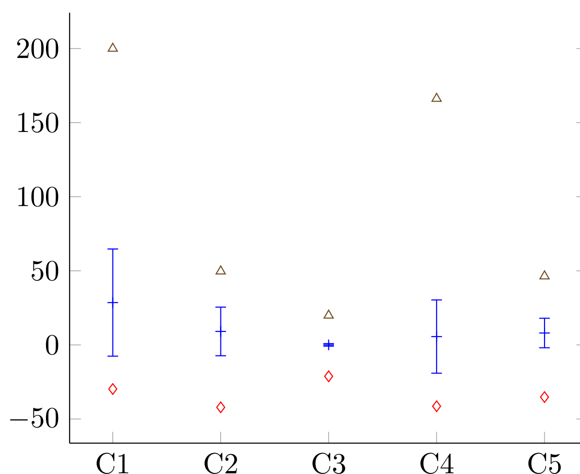


Figure 5.5: Average mean and standard deviation of root dual bound improvement of our branch-price-and-cut approach as compared to CPLEX in percent. The best bound over all tolerances has been used. Triangles show the best obtained improvement (e.g., 200% in C1) and diamonds show the worst lower bound (e.g., 44% worse than CPLEX in C2).

precedence cuts in our model by evaluating the number of branch-and-bound nodes and the running time. In Figures 5.6 and 5.7 the first bar of each calendar (black) gives the number of solved instances obtained by CPLEX, the second (white) bar represents this number for the branch-and-price approach without precedence cuts. The first five grey bars symbolize the results for setting "τ_{greedy}" in increasing order of tolerances and the last five bars stand for settings "τ_{best}" in increasing order of tolerances.

Figure 5.6 shows for the instance set N50E70 that the pure branch-and-price algorithm without precedence cuts outperforms CPLEX and adding precedence cuts seems

to be a bad idea. The set N50E70 with fewer precedence constraints than N50E100 (see Figure 5.7) is the harder one, as expected. In some cases the pure MIP approach is even better than the branch-and-price approach with additional precedence cuts. This is because the time spent for separating new cuts, pricing new variables, and the additional LP iterations lead to too many timeouts and is therefore not competitive. Figure 5.6 clearly shows that a high tolerance (tol.) leads to better results, because fewer (and stronger) precedence cuts are added. Hence, not using any precedence cuts at all is, at first sight, a good decision.

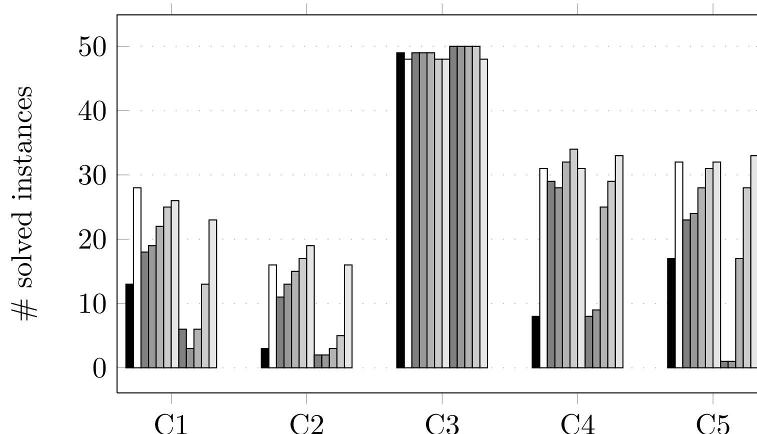


Figure 5.6: Number of solved instances for N50E70 with cuts already separated in the root node; each set of bars from left to right corresponds to rows in Table 5.1. A CPLEX run on the standard MIP is black; our branch-and-price algorithm without cuts is white; and the two groups of different shades of grey show runs of the full branch-and-price algorithm with precedence cuts enabled, with settings τ_{greedy} and τ_{best}, respectively, with increasing tolerances (lighter grey is larger tolerance).

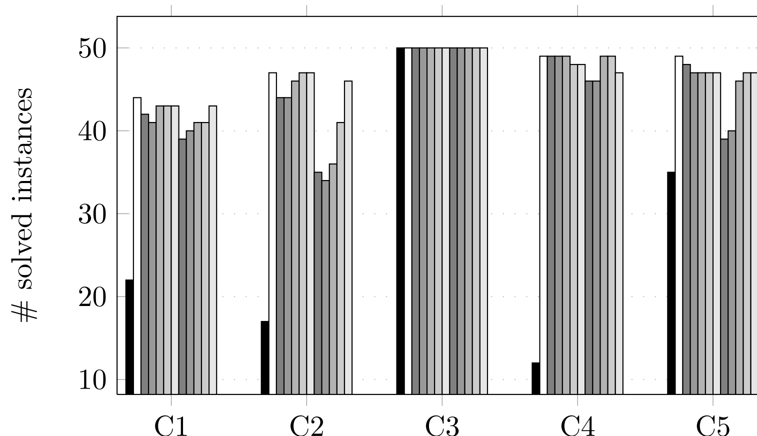


Figure 5.7: Number of solved instances for N50E100 with cuts already separated in the root node.

Nevertheless, it is possible to increase the root dual bound on several instances by using precedence cuts (5.25). During root solving most jobs are able to slide in their time window and the precedence cuts result in more binary configuration variables that are

smeared over the time horizon. Hence, these cuts should not be used in the root node but still might be beneficial to prune certain nodes of the branch-and-bound tree. Recall that our branching scheme first branches on the resource capacities R_k and afterwards on the start and completion time variables. Fixed resource capacities in a node already decide on a lot of structure for the scheduling problem, since several modes of a job may no longer be valid. Thus, this seems to be a good point to separate precedence cuts (5.25). Figures 5.8 and 5.9 show that more instances than before can be solved using the precedence cuts. In several cases a tolerance of 10^{-4} belongs to the best choices for the separation procedure. Especially, for the harder instances N50E70 some more instances of each calendar test set can be solved in the time limit to proven optimality in contrast to CPLEX or a setting without additional precedence cuts.

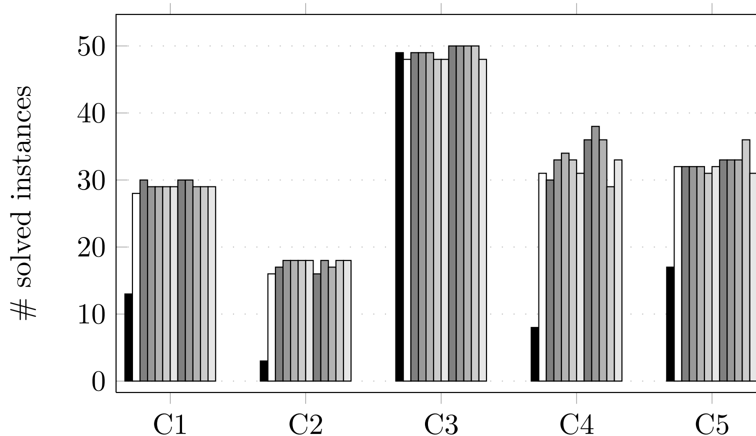


Figure 5.8: Number of solved instances of test set N50E70 if cuts are separated after resource capacity variables are fixed.

Next, we elaborate on the solving time and on the number of nodes needed to find an optimal solution and prove its optimality. The results on the total number of solved instances showed that it is beneficial to separate precedence cuts after resource variables are fixed.

Tables 5.1 and 5.2 reveal that using additional precedence cuts enables us to decrease the number of tree nodes by 10%–20% on average. For several hard instances, e.g., in calendar C5 in test set N50E70 a decrease by even 50% is possible. Best results in terms of nodes are obtained when τ_{best} is computed. Nevertheless, this does not carry over to a reduced running time. For C1 the running times increase (τ_{best} vs. τ_{greedy}), whereas for C5 it decreases and the running time is about 10% faster than if no precedence cuts are separated. That is, there is the usual trade-off between quality and time, and cutting planes may be most interesting in memory critical applications.

A tolerance between 0.1 and 0.3 gives the overall best results as higher and lower tolerances usually increase the running times.

Table 5.3 shows that using precedence cuts, not only increases the number of best lower bounds that are found but this way often better primal solutions are found, e.g., on calendar C4 where the number of best primal bounds is increased from 39 to 45 with precedence cuts.

While Tables 5.1 and 5.2 only show a slight improvement considering the average running time and the number of nodes needed (in the shifted geometric mean) if prece-

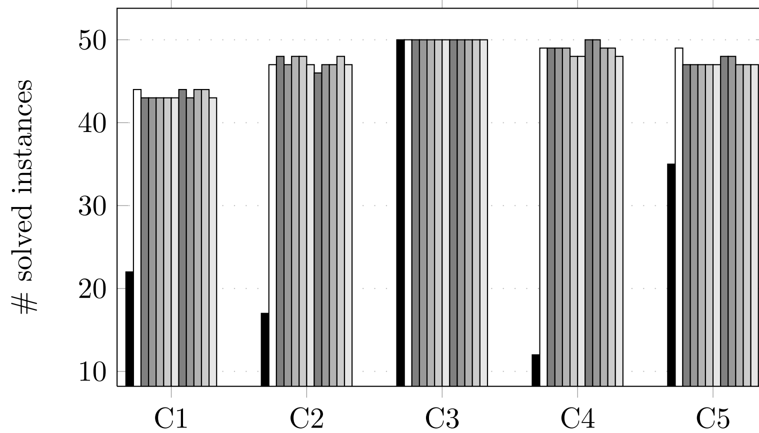


Figure 5.9: Number of solved instances of test set N50E100 if cuts are separated after resource capacity variables are fixed.

Table 5.1: Comparison of tree nodes and time for N50E70 – cuts are only separated after resources are fixed. Means are computed over the instances solved by all settings, of which there are 27, 14, 48, 26, and 31 for Calendars C1 to C5.

		C1		C2		C3		C4		C5	
	tol.	nodes	time	nodes	time	nodes	time	nodes	time	nodes	time
no cuts		12.0	660.7	10.7	546.3	1.0	6.4	11.4	200.5	21.3	356.2
	0.0	10.0	667.9	9.5	553.4	1.0	6.4	8.9	201.0	14.3	346.3
	0.1	10.2	662.7	8.8	518.7	1.0	6.4	8.3	197.2	16.7	361.8
τ_{greedy}	0.3	10.6	632.6	9.0	523.4	1.0	6.4	10.0	204.2	15.9	338.9
	0.5	11.4	661.0	10.5	538.9	1.0	6.4	11.2	216.9	19.1	347.2
	0.8	11.4	611.5	10.7	560.7	1.0	6.3	11.1	200.4	19.7	351.5
	0.0	9.2	704.6	8.3	536.0	1.0	6.4	7.4	196.4	12.0	353.6
	0.1	8.9	673.1	8.4	559.8	1.0	6.4	7.9	202.8	11.6	341.3
τ_{best}	0.3	9.2	653.6	8.6	542.7	1.0	6.4	7.7	197.6	10.9	323.3
	0.5	10.1	660.7	9.8	560.6	1.0	6.4	9.1	201.1	15.6	337.6
	0.8	11.1	608.4	10.3	528.2	1.0	6.3	12.4	212.4	16.6	338.9

Table 5.2: Comparison of tree nodes and time for N50.E100 – cuts are only separated after resources are fixed. Means are computed over the instances solved by all settings, of which there are 42, 44, 50, 48, and 47 for Calendars C1 to C5 respectively.

		C1		C2		C3		C4		C5	
	tol.	nodes	time	nodes	time	nodes	time	nodes	time	nodes	time
no cuts		5.3	31.0	9.8	88.8	1.1	1.0	8.4	18.1	8.2	23.8
	0.0	5.4	32.6	8.0	83.6	1.1	1.0	6.3	16.4	7.7	25.5
	0.1	4.8	31.1	8.2	86.9	1.1	1.0	6.4	16.5	7.3	24.3
τ_{greedy}	0.3	5.4	30.5	9.1	87.3	1.1	1.0	6.5	16.3	7.7	24.1
	0.5	5.1	30.7	9.1	85.5	1.1	1.0	6.8	16.5	8.2	24.0
	0.8	5.4	29.7	10.3	88.1	1.1	0.9	7.4	16.4	9.0	23.6
	0.0	4.8	32.3	7.5	88.8	1.1	1.0	5.9	16.5	6.3	23.5
	0.1	4.8	32.5	7.2	86.8	1.1	1.0	5.7	16.4	6.2	23.0
τ_{best}	0.3	4.7	30.5	7.6	87.3	1.1	1.0	5.8	16.3	6.5	22.9
	0.5	5.1	31.4	7.7	83.2	1.1	1.0	6.4	16.5	7.8	24.0
	0.8	5.4	29.8	10.0	92.2	1.1	0.9	7.2	16.2	8.5	23.3

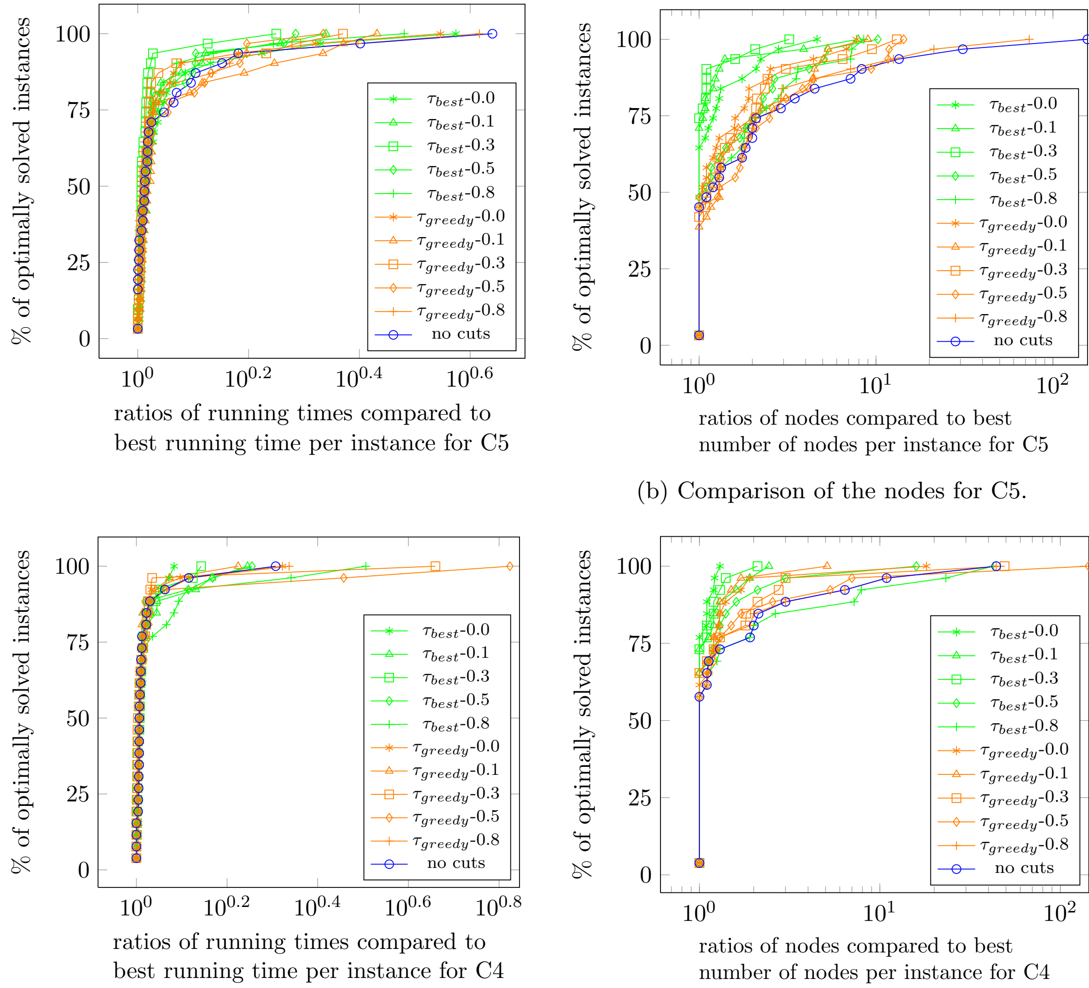
dence inequalities are separated after resource capacities are fixed, Figure 5.10 gives a more detailed comparison by comparing the ratios of the number of nodes (running time) needed per instance compared to the best running time by any of the settings listed in the

Table 5.3: The number of best lower and best upper bounds that were found throughout search are displayed. for N50.E70 and below N50.E100. Cuts are separated after resource variables are fixed. “nS” is the number of optimally solved instances, “bLB” (“bUB”) denotes how often the best lower (upper) bound is found.

		C1			C2			C3			C4			C5		
	tol.	nS	bLB	bUB	nS	bLB	bUB	nS	bLB	bUB	nS	bLB	bUB	nS	bLB	bUB
no cuts		28	45	48	16	48	39	48	48	50	31	48	39	32	47	43
τ_{greedy}	0.0	30	45	49	17	48	41	49	49	50	30	49	37	32	47	41
	0.1	29	46	49	18	49	43	49	49	50	33	49	40	32	46	42
	0.3	29	46	49	18	49	40	49	49	50	34	49	42	32	46	43
	0.5	29	46	49	18	49	42	48	48	50	33	49	40	31	46	44
	0.8	29	47	49	18	49	42	48	48	50	31	48	39	32	48	43
τ_{best}	0.0	30	45	50	16	48	40	50	50	50	36	50	43	33	49	43
	0.1	30	47	48	18	47	41	50	50	50	38	49	44	33	49	44
	0.3	29	47	49	17	49	40	50	50	50	36	49	45	33	49	44
	0.5	29	47	49	18	49	42	50	50	50	29	50	38	36	49	46
	0.8	29	47	49	18	48	40	48	48	50	33	48	41	31	48	43

		C1			C2			C3			C4			C5		
	tol.	nS	bLB	bUB	nS	bLB	bUB	nS	bLB	bUB	nS	bLB	bUB	nS	bLB	bUB
no cuts		44	48	50	47	49	50	50	50	50	49	49	50	49	49	50
τ_{greedy}	0.0	43	47	48	48	50	50	50	50	50	49	49	50	47	47	50
	0.1	43	46	49	47	50	49	50	50	50	49	49	50	47	47	50
	0.3	43	46	49	48	50	50	50	50	50	49	49	50	47	47	50
	0.5	43	46	50	48	50	50	50	50	50	48	48	50	47	47	49
	0.8	43	46	50	47	49	50	50	50	50	48	48	50	47	47	49
τ_{best}	0.0	44	48	48	46	49	49	50	50	50	50	50	50	48	48	50
	0.1	43	47	48	47	50	49	50	50	50	50	50	50	48	48	50
	0.3	44	47	50	47	50	48	50	50	50	49	49	50	47	47	50
	0.5	44	47	49	48	50	50	50	50	50	49	49	50	47	47	50
	0.8	43	46	50	47	49	50	50	50	50	48	48	50	47	47	49

tables. We see that, e.g., for calendar C5 on more than 50% of the instances the precedence inequalities remarkably reduce the number of nodes needed. On several instances, the setting without precedence inequalities needs between ten and 100 times more nodes than the best setting with precedence inequalities. But we also see that the reduction in terms of running times is much smaller as separating these cuts and computing the new objective coefficients in the pricing problems is costly, too. Similarly, for calendar C4 we observe a decrease in the number of nodes needed, whereas the reduction in terms of running time is rather modest and can only be seen on less than 20 instances. We do not elaborate on the results for calendars C1 to C3 here. On calendars C1 and C2, results are similar to C5, whereas on C3 not much changes, as on these instances the dual bounds have not been a bottleneck.



(b) Comparison of the nodes for C5.

Figure 5.10: Ratios of nodes and running times for all optimally solved instances from N50E70 for calendar C5 on top and for calendar C4 below.

Conclusion and outlook

The Turnaround Scheduling problem is complex and interesting from a computational point of view. The decomposition approach invented here builds on subschedules per working shift and hopefully enables further research on that topic. The strength of the better dual bounds derived after applying Dantzig-Wolfe decomposition calls for being tested in further scheduling applications and extensions, maybe to net present value problems as studied in Section 4.1. Even the involved $\mathcal{NP}$-hard pricing problems do not become a bottleneck at all, as the size of these problems is well controlled and generic MIP solvers can be used to generate new variables.

Our computational study shows that decomposing the problem resource-wise per availability period helps to solve instances with 50 multi-mode jobs and a large planning horizon, whereas in benchmark instances from the literature, instances with 30 multi-mode jobs cannot be solved to optimality. CPLEX also fails on more than half of our instances. Here, the availability periods may also make the problem easier after

decomposition.

Our algorithm is generic and components like the pricing problem may also be solved via constraint programming algorithms or partially with a heuristic. We believe that the general approach is well-suited for similar problems, in particular, when the objective function is “complicated.”

It will be worth-while to extend all CP techniques developed in Sections 2.1 and 3 and to apply them to large-scale instances of the Turnaround Scheduling problem. Our preliminary work in this direction suffers from weak propagation as the demands and processing times per job (mode) must be handled carefully. Memory issues in SCIP from the huge amount of variables further complicated these experiments. As SAT solvers become faster also on scheduling problems, it can be expected that their use for Turnaround Scheduling models is also a promising research direction. But as the objective function may become more complex, our work in this chapter shows that sophisticated IP techniques are in particular useful to strengthen the lower bounds which may not easily be the case in CP or SAT approaches.

Bibliography

[1] T. Achterberg. Conflict analysis in mixed integer programming. *Discrete Optimization*, 4(1):4–20, 2007. Special issue: Mixed Integer Programming.

[2] T. Achterberg. SCIP: solving constraint integer programs. *Mathematic Programming Computation*, 1(1):1–41, 2009.

[3] T. Achterberg and T. Berthold. Hybrid branching. In *Proceedings of the CPAIOR 2009*, volume 5547 of *LNCS*, pages 309–311, 2009.

[4] S.M. Adel and S.E. Elmaghraby. Optimal linear approximation in project compression. *IIE Transaction*, 16(4):339–347, 1984.

[5] 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.

[6] R. Alvarez-Valdés and J.M. Tamarit. The project scheduling polyhedron: Dimension, facets and lifting theorems. *European Journal of Operational Research*, 67(2):204–220, 1993.

[7] K. Andersen and Y. Pochet. Coefficient strengthening: A tool for reformulating mixed-integer programs. *Mathematical Programming*, 122:121–154, 2010.

[8] C. Ansótegui, M. Bofill, M. Palahí, J. Suy, and M. Villaret. Satisfiability Modulo Theories: An efficient approach for the Resource-Constrained Project Scheduling Problem. In R. Barták and M. Milano, editors, *Proceedings of the Ninth Symposium on Abstraction, Reformulation, and Approximation, SARA 2011*, pages 2–9. AAAI, 2011.

[9] D. Applegate, R. Bixby, V. Chvátal, and W. Cook. Finding cuts in the tsp (a preliminary report), 1994.

[10] E.M. Arkin and E.B. Silverberg. Scheduling jobs with fixed start and end times. *Discrete Applied Mathematics*, 18:1–8, 1987.

[11] I. Aron, J.N. Hooker, and T.H. Yunes. SIMPL: A system for integrating optimization techniques. In *Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2004), LNCS 3011*, pages 21–36, 2004.

[12] C. Artigues, S. Demassey, and E. Neron. *Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications.* ISTE, 2007.

[13] C. Artigues, P. Michelon, and S. Reusser. Insertion techniques for static and dynamic Resource-Constrained Project Scheduling. *European Journal of Operational Research*, 149:249–267, 2003.

[14] K.R. Baker. *Introduction to Sequencing and Scheduling.* John Wiley & Sons, New York, 1974.

[15] M. Bandelloni, M. Tucci, and R. Rinaldi. Optimal resource leveling using non-serial dynamic programming. *Journal of the Society for Industrial and Applied Mathematics*, 78(2):162–177, 1994.

[16] N. Bansal, A. Chakrabarti, A. Epstein, and B. Schieber. A quasi-PTAS for unsplittable flow on line graphs. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, STOC '06, pages 721–729, New York, USA, 2006. ACM.

[17] N. Bansal, Z. Friggstad, R. Khandekar, and M. Salavatipour. A logarithmic approximation for unsplittable flow on line graphs. In *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '09, pages 702–709, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics.

[18] P. Baptiste, P. Brucker, M. Chrobak, C. Dürr, S.A. Kravchenko, and F. Sourd. The complexity of mean flow time scheduling problems with release times. *Journal of Scheduling*, 10:139–146, 2007.

[19] P. Baptiste and S. Demassey. Tight LP bounds for Resource-Constrained Project Scheduling. *OR Spectrum*, 26(2):251–262, 2004.

[20] P. Baptiste, A. Jouglet, and D. Savourey. Lower bounds for parallel machine scheduling problems. *International Journal of Operational Research*, 3(6):643–664, 2011.

[21] P. Baptiste and C. Le Pape. Constraint propagation and decomposition techniques for highly disjunctive and highly cumulative project scheduling problems. *Constraints*, 5(1/2):119–139, 2000.

[22] P. Baptiste, C. Le Pape, and W. Nuijten. *Constraint-based scheduling: Applying constraint programming to scheduling problems*. International Series in Operations Research & Management Science, 39. Kluwer Academic Publishers, Boston, MA, 2001.

[23] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, and B. Schieber. A unified approach to approximating resource allocation and scheduling. In *Proceedings of the thirty-second Annual ACM Symposium on Theory of Computing*, STOC '00, pages 735–744, New York, NY, USA, 2000. ACM.

[24] R. Bar-Yehuda, M. Beder, Y. Cohen, and D. Rawitz. Resource allocation in bounded degree trees. In *Proceedings of the 14th Annual European Symposium Algorithms, ESA 2006*, volume 4168 of *LNCS*, pages 64–75. Springer-Verlag, 2006.

[25] S.M. Baroum and J.H. Patterson. *An exact solution procedure for maximizing the net present value of cash flows in a network*, pages 107–134. J. Weglarz, Kluwer Academic Publishers, Boston, MA, 1990.

[26] M. Bartlett, A.M. Frisch, Y. Hamadi, I. Miguel, A. Tarim, and C. Unsworth. The temporal knapsack problem and its solution. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 3524 of *LNCS*, pages 34–48. Springer-Verlag, 2005.

[27] M. Bartusch, R. H. Möhring, and F. J. Radermacher. Scheduling project networks with resource constraints and time windows. *Annals of Operations Research*, 16(1-4):201–240, 1988.

[28] J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Computational Management Science*, 2:3–19, 2005.

[29] T. Berthold, S. Heinz, M.E. Lübbecke, R.H. Möhring, and J. Schulz. A Constraint Integer Programming approach for Resource-Constrained Project Scheduling. In A. Lodi, M. Milano, and P. Toth, editors, *Proceedings of the 7th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2010)*, volume 6140 of *LNCS*, pages 313–317. Springer, 2010.

[30] T. Berthold, S. Heinz, and J. Schulz. An approximative criterion for the potential of energetic reasoning. In A. Marchetti-Spaccamela and M. Segal, editors, *Proceedings of the First International ICST Conference on Theory and Practice of Algorithms in (Computer) Systems (TAPAS 2011)*, volume 6595 of *LNCS*, pages 229–239. Springer, 2011.

[31] L. Bianco and M. Caramia. Minimizing the completion time of a project under resource constraints and feeding precedence relations: A lagrangian relaxation based lower bound. *4OR: A Quarterly Journal of Operations Research*, pages 1–19, 2011.

[32] L. Bianco and M. Caramia. A new lower bound for the Resource-Constrained Project Scheduling Problem with generalized precedence relations. *Computers and Operations Research*, 38(1):14–20, 2011.

[33] 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.

[34] P. Bonsma, J. Schulz, and A. Wiese. A constant factor approximation algorithm for unsplittable flow on paths. *CoRR*, abs/1102.3643, 2011.

[35] A. Bourges and J. Killebrew. Variation in activity level in a cyclical arrow diagram. *Journal of Industrial Engineering*, 13(2):76–83, 1962.

[36] P. Brucker, A. Drexl, R. Möhring, K. Neumann, and E. Pesch. Resource-Constrained Project Scheduling: Notation, Classification, Models, and Methods. *European Journal of Operational Research*, 112:3–41, 1999.

[37] P. Brucker and S. Knust. A linear programming and constraint propagation-based lower bound for the RCPSP. *European Journal of Operational Research*, 127(2):355–362, 2000.

[38] P. Brucker and S. Knust. Resource-Constrained Project Scheduling. In *Complex Scheduling*, GOR-Publications, pages 117–238. Springer, 2012.

[39] P. Brucker, S. Knust, A. Schoo, and O. Thiele. A branch and bound algorithm for the Resource-Constrained Project Scheduling Problem. *European Journal of Operational Research*, 107(2):272–288, 1998.

[40] G. Calinescu, A. Chakrabarti, H.J. Karloff, and Y. Rabani. Improved approximation algorithms for resource allocation. In *Proceedings of the 9th International Conference on Integer Programming and Combinatorial Optimization*, IPCO '09, pages 401–414, London, UK, 2002. Springer-Verlag.

[41] J. Carlier. The one-machine sequencing problem. *European Journal of Operational Research*, 11(1):42–47, 1982.

[42] J. Carlier and E. Néron. A new LP-based lower bound for the cumulative scheduling problem. *European Journal of Operational Research*, 127(2):363–382, 2000.

[43] J. Carlier and E. Néron. On linear lower bounds for the Resource-Constrained Project Scheduling Problem. *European Journal of Operational Research*, 149(2):314–324, 2003.

[44] J. Carlier and E. Néron. Computing redundant resources for the Resource-Constrained Project Scheduling Problem. *European Journal of Operational Research*, 176(3):1452–1463, 2007.

[45] J. Carlier and E. Pinson. Jackson's pseudo-preemptive schedule and cumulative scheduling problems. *Discrete Applied Mathematics*, 145:80–94, 2004.

[46] R.D. Carr, L.K. Fleischer, V.J. Leung, and C.A. Phillips. Strengthening integrality gaps for capacitated network design and covering problems. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, SODA '00, pages 106–115. SIAM, 2000.

[47] Y. Caseau and F. Laburthe. A constraint based approach to the RCPSP. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP 1997)*, volume 1330 of *LNCS*. Springer, 1997.

[48] C.C.B. Cavalcante, 1997. `www.dcc.unicamp.br/~cid/SPLC/SPLC.html`.

[49] C.C.B. Cavalcante, V.C. Cavalcante, C.C. Ribeiro, and C.C. De Souza. *Parallel Cooperative Approaches for the Labor Constrained Scheduling Problem*. Kluwer, 2000.

[50] C.C.B. Cavalcante, C.C. De Souza, M.W.P. Savelsbergh, Y. Wang, and L.A. Wolsey. Scheduling projects with labor constraints. *Discrete Applied Mathematics*, 112:27–52, 2001.

[51] V. Chakaravarthy, A. Kumar, S. Roy, and Y. Sabharwal. Resource allocation for covering time varying demands. In *Algorithms – ESA 2011*, volume 6942 of *LNCS*, pages 543–554. Springer, 2011.

[52] V.T. Chakaravarthy, G.R. Parija, S. Roy, Y. Sabharwal, and A. Kumar. Minimum cost resource allocation for meeting job requirements. *Parallel and Distributed Processing Symposium, International*, pages 14–23, 2011.

[53] A. Chakrabarti, C. Chekuri, A. Gupta, and A. Kumar. Approximation algorithms for the unsplittable flow problem. *Algorithmica*, pages 53–78, 2007.

[54] C. Chekuri, A. Ene, and N. Korula. Unsplittable flow in paths and trees and column-restricted packing integer programs. In *Proceedings of the 12th International Workshop and 13th International Workshop on Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, APPROX '09 / RANDOM '09, pages 42–55. Springer, 2009.

[55] C. Chekuri, M. Mydlarz, and F. Shepherd. Multicommodity demand flow in a tree and packing integer programs. *ACM Transactions on Algorithms*, 3:27, 2007.

[56] B. Chen, R. Hassin, and M. Tzur. Allocation of bandwidth and storage. *IIE Transactions*, 34:501–507, 2002.

[57] W.-N. Chen, J. Zhang, H.S.-H. Chung, R.-Z. Huang, and O. Liu. Optimizing discounted cash flows in project scheduling: an ant colony optimization approach. *Transactions on Systems, Man and Cybernetics – Part C*, 40:64–77, January 2010.

[58] Choco. http://www.emn.fr/z-info/choco-solver/.

[59] N. Christofides, R. Alvarez-Valdés, and J. M. Tamarit. Project scheduling with resource constraints: A branch and bound approach. *European Journal of Operational Research*, 29(3):262–273, 1987.

[60] M. Chrobak, G. Woeginger, K. Makino, and H. Xu. Caching is hard even in the fault model. In M. de Berg and U. Meyer, editors, *Algorithms ESA 2010*, volume 6346 of *LNCS*, pages 195–206. Springer, 2010.

[61] V. Chvátal. Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete Mathematics*, 4(4):305–337, 1973.

[62] M. Cieliebak, T. Erlebach, F. Hennecke, B. Weber, and P. Widmayer. Scheduling with release times and deadlines on a minimum number of machines. In J.-J. Lévy, E.W. Mayr, and J.C. Mitchell, editors, *IFIP TCS*, pages 209–222. Kluwer, 2004.

[63] Comet. http://dynadec.com/.

[64] S.A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third Annual ACM Symposium on Theory of Computing*, STOC '71, pages 151–158. ACM, 1971.

[65] M. C. Cooper. An optimal k-consistency algorithm. *Artificial Intelligence*, 41:89–95, 1989.

[66] G. Cornuéjols. Valid inequalities for mixed integer linear programs. *Mathematical Programming*, 112:3–44, 2008.

[67] E.T. Coughlan, M.E. Lübbecke, and J. Schulz. A branch-and-price algorithm for multi-mode resource leveling. In P. Festa, editor, *SEA*, volume 6049 of *LNCS*, pages 226–238. Springer, 2010.

[68] IBM ILOG CPLEX. http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/.

[69] G. Dantzig. *Linear Programming and Extensions.* Princeton University Press, 1998.

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

[71] G.B. Dantzig and P. Wolfe. The decomposition algorithm for linear programs. *Econometrica*, 29(4):767–778, 1961.

[72] A. Darmann, U. Pferschy, and J. Schauer. Resource allocation with time intervals. *Theoretical Computer Science*, 411:4217–4234, 2010.

[73] E. Davis. Constraint propagation with interval labels. *Artificial Intelligence*, 32:281–331, 1987.

[74] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5:394–397, 1962.

[75] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960.

[76] P. De, E.J. Dunne, J.B. Ghosh, and C.E. Wells. The discrete time-cost tradeoff problem revisited. *European Journal of Operational Research*, 81:225–238, 1995.

[77] P. De, J.E. Dunne, J.B. Ghosh, and C.E. Wells. Complexity of the discrete time-cost tradeoff problem for project networks. *Operations Research*, 45(2):302–306, 1997.

[78] B. De Reyck and W. Herroelen. Assembly line balancing by Resource-Constrained Project Scheduling: A critical appraisal. *Foundations of Computing and Control Engineering*, 22(3):143–167, 1997.

[79] V.G. Deineko and G.J. Woeginger. Hardness of approximation of the discrete time-cost tradeoff problem. *Operations Research Letters*, 29(5):207–210, 2001.

[80] S. Demassey, C. Artigues, P. Baptiste, and P. Michelon. Lagrangean relaxation-based lower bounds for the RCPSP. In *9th International Workshop on Project Management and Scheduling (PMS'04)*, 2004.

[81] S. Demassey, C. Artigues, and P. Michelon. Constraint-propagation-based cutting planes: An application to the Resource-Constrained Project Scheduling Problem. *INFORMS Journal on Computing*, 17(1):52–65, 2005.

[82] E. Demeulemeester. Minimizing resource availability costs in time-limited project networks. *Management Science*, 41:1590–1598, 1995.

[83] E. Demeulemeester and W. Herroelen. A branch-and-bound procedure for the multiple Resource-Constrained Project Scheduling Problem. *Management Science*, 38(12):1803–1818, 1992.

[84] E. Demeulemeester and W. Herroelen. New benchmark results for the Resource-Constrained Project Scheduling Problem. *Management Science*, 43(11):1485–1492, 1997.

[85] E. Demeulemeester and W. Herroelen. *Project Scheduling: A Research Handbook.* Kluwer, 2002.

[86] E. Demeulemeester, W. Herroelen, and P. Van Dommelen. An optimal recursive search procedure for the deterministic unconstrained max-npv project scheduling problem. Technical report, Katholieke University Leuven, 1996.

[87] J. Desrosiers and M.E. Lübbecke. A primer in column generation. In G. Desaulniers, J. Desrosiers, and M.M. Solomon, editors, *Column Generation*, pages 1–32. Springer, 2005.

[88] J. Desrosiers and M.E. Lübbecke. Branch-price-and-cut algorithms. In J.J. Cochran, editor, *Encyclopedia of Operations Research and Management Science.* PUWILEY, 2011.

[89] B.L. Dietrich, L.F. Escudero, and F. Chance. Efficient reformulation for 0-1 programs: Methods and computational results. *Discrete Applied Mathematics*, 42(2-3):147–175, 1993.

[90] M. Dincbas, P. Van Hentenryck, H. Simonis, A. Aggoun, T. Graf, and F. Berthier. The constraint logic programming language CHIP. In *FGCS*, pages 693–702, 1988.

[91] R.H. Doersch and J.H. Patterson. Scheduling a project to maximize its present value: A zero-one programming approach. *Management Science*, 23(8):882–889, 1977.

[92] U. Dorndorf, E. Pesch, and T. Phan Huy. Constraint propagation techniques for the disjunctive scheduling problem. *Artificial Intelligence*, 122(1-2):189–240, 2000.

[93] U. Dorndorf, E. Pesch, and T. Phan-Huy. A branch-and-bound algorithm for the Resource-Constrained Project Scheduling Problem. *Mathematical Methods of Operations Research*, 52:413–439, 2000.

[94] U. Dorndorf, E. Pesch, and T. Phan-Huy. Constraint propagation and problem decomposition: A preprocessing procedure for the job shop problem. *Annals of Operations Research*, 115:125–145, 2002.

[95] U. Dorndorf, T. Phan-Huy, and E. Pesch. *A survey of interval capacity consistency tests for time- and resource-constrained scheduling*, chapter 10, pages 213–238. J. Weglarz, Kluwer Academic, Boston, MA, 1999.

[96] A. Drexl and A. Kimms. Optimization guided lower and upper bounds for the resource investment problem. *The Journal of the Operational Research Society*, 52(3):340–351, 2001.

[97] M. Drozdowski and P. Dell' Olmo. Scheduling multiprocessor tasks for mean flow time criterion. *Computers & Operations Research*, 27(6):571–585, 2000.

[98] J. Du and J.Y.-T. Leung. Complexity of scheduling parallel task systems. *SIAM Journal on Discrete Mathematics*, 2(4):473–487, 1989.

[99] J. Du, J.Y.-T. Leung, and G.H. Young. Minimizing mean flow time with release time constraint. *Theoretical Computer Science*, 75(3):347–355, 1990.

[100] J.Z. Du and J.Y.-T. Leung. Minimizing mean flow time with release time and deadline constraints. *Journal of Algorithms*, 14(1):45–68, 1993.

[101] C. W. Duin and E. Van Sluis. On the complexity of adjacent resource scheduling. *Journal of Scheduling*, 9:49–62, 2006.

[102] S.M. Easa. Resource leveling in construction by optimization. *Journal of Construction Engineering and Management*, 115(2):302–316, 1989.

[103] N. Eén and N. Sörensson. MiniSAT. http://minisat.se/MiniSat.html.

[104] N. Eén and N. Sörensson. An extensible SAT-solver. In *Theory and Applications of Satisfiability Testing*, volume 2919 of *LNCS*, pages 333–336. Springer, 2004.

[105] S.E. Elmaghraby and W.S. Herroelen. The scheduling of activities to maximize the net present value of projects. *European Journal of Operational Research*, 49(1):35–49, 1990. Project Management and Scheduling.

[106] S.E. Elmaghraby and J. Kamburowski. The analysis of activity networks under generalized precedence relations (gprs). *Management Science*, 38(9):1245–1263, 1992.

[107] L. Escudero, S. Martello, and P. Toth. On tightening 0-1 programs based on extensions of pure 0-1 knapsack and subset-sum problems. *Annals of Operations Research*, 81:379–404, 1998.

[108] G. Even, M.M. Halldórsson, L. Kaplan, and D. Ron. Scheduling with conflicts: online and offline algorithms. *Journal of Scheduling*, 12(2):199–224, 2009.

[109] J.E. Falk and J.L. Horowitz. Critical path problems with concave cost-time curves. *Management Science*, 19:446–455, 1972.

[110] U. Feige and J. Kilian. Zero knowledge and the chromatic number. *Journal of Computer and System Sciences*, 57(2):187–199, 1998.

[111] M. Fischetti and A. Lodi. Mipping closures: An instant survey. *Graphs and Combinatorics*, 23:233–243, 2007.

[112] B. Franck, K. Neumann, and C. Schwindt. Project scheduling with calendars. *OR Spektrum*, 23:325–334, 2001.

[113] E.C. Freuder. Synthesizing constraint expressions. *Commun. ACM*, 21(11):958–966, 1978.

[114] D.R. Fulkerson. A network flow computation for project cost curves. *Management Science*, 7:167–178, 1961.

[115] E.R. Gafarov, A.A. Lazarev, and F. Werner. Approximability results for the Resource-Constrained Project Scheduling Problem with a single type of resources. *Annals of Operations Research*, pages 1–16, 2012.

[116] M.R. Garey and D.S. Johnson. Complexity results for multiprocessor scheduling under resource constraints. *SIAM Journal on Computing*, 4(4):397–411, 1975.

[117] M.R. Garey and D.S. Johnson. *Computers and intractability.* W. H. Freeman and Co., San Francisco, Calif., 1979. A guide to the theory of NP-completeness, A Series of Books in the Mathematical Sciences.

[118] T. Gather, J. Zimmermann, and J.-H. Bartels. Exact methods for the resource levelling problem. *Journal of Scheduling*, pages 1–13, 2010.

[119] GECODE. Generic constraint development environment. http://www.gecode.org/.

[120] M.L. Ginsberg and D.A. McAllester. GSAT and dynamic backtracking. *Journal of Artificial Intelligence Research*, 1:25–46, 1994.

[121] F. Glover, M. Laguna, R. Mart, and K. Womer. Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 39:653–684, 2000.

[122] M.X. Goemans. A supermodular relaxation for scheduling with release dates. In W.H. Cunningham, S.T. McCormick, and M. Queyranne, editors, *IPCO*, volume 1084 of *LNCS*, pages 288–300. Springer, 1996.

[123] M.X. Goemans. Improved approximation algorthims for scheduling with release dates. In *Proceedings of the eighth annual ACM-SIAM Symposium on Discrete Algorithms (SODA 1997)*, pages 591–598. SIAM, 1997.

[124] M.X. Goemans, M. Queyranne, A.S. Schulz, M. Skutella, and Y. Wang. Single machine scheduling with release dates. *SIAM Journal on Discrete Mathematics*, 15(2):165–192, 2002.

[125] R.E. Gomory. Outline of an algorithm for integer solutions to linear program. *Bulletin of the American Mathematical Society*, 64(5):275–278, 1958.

[126] E. Goto, T. Joko, K. Fujisawa, N. Katoh, and S. Furusaka. Maximizing net present value for generalized Resource-Constrained Project Scheduling Problem, 2000. Working paper Nomura Research Institute, Japan.

[127] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 4:287–326, 1979.

[128] A. Grigoriev, M. Sviridenko, and M. Uetz. Machine scheduling with resource dependent processing times. *Mathematical Programming*, 110(1):209–228, 2007.

[129] R.C. Grinold. The payment scheduling problem. *Naval Research Logistics Quarterly*, 19(1):123–136, 1972.

[130] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization.* Springer, 1988.

[131] E. Günther, , F.G. König, and N. Megow. Scheduling and packing malleable tasks with precedence constraints of bounded width. In *7th International Workshop on Approximation and Online Algorithms (WAOA 2009)*, LNCS, pages 170–181. Springer, 2010.

[132] R.B. Harris. Packing method for resource leveling (pack). *Journal of Construction Engineering and Management*, 116(2):331–350, 1990.

[133] S. Hartmann. A competitive genetic algorithm for Resource-Constrained Project Scheduling. *Naval Research Logistics (NRL)*, 45(7):733–750, 1998.

[134] S. Hartmann. Project scheduling with multiple modes: A genetic algorithm. *Annals of Operations Research*, 102(1-4):111–135, 2001.

[135] 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.

[136] S. Hartmann and R. Kolisch. Experimental evaluation of state-of-the-art heuristics for the Resource-Constrained Project Scheduling Problem. *European Journal of Operational Research*, 127(2):394–407, 2000.

[137] S. Heinz and J. Schulz. Explanations for the cumulative constraint: An experimental study. In P.M. Pardalos and S. Rebennack, editors, *Experimental Algorithms (SEA 2011)*, volume 6630 of *LNCS*, pages 400–409. Springer, 2011.

[138] S. Heipcke and Y. Colombani. A new constraint programming approach to large scale resource constrained scheduling. Workshop on Models and Algorithms for Planning and Scheduling Problems (MAPSP 1997), Cambridge, UK, 1997.

[139] S. Heipcke, Y. Colombani, C.C.B. Cavalcante, and C.C. de Souza. Scheduling under labour resource constraints. *Constraints*, 5:415–422, 2000.

[140] W. Herroelen, B. De Reyck, and E. Demeuleemester. A classification scheme for project scheduling problems. In J. Weglarz, editor, *Project Scheduling – Recent Models, Algorithms and Applications*, pages 1–26. Kluwer Academic Publishers, 1998.

[141] L. Hidri, A. Gharbi, and M. Haouari. Energetic reasoning revisited: Application to parallel machine scheduling. *Journal of Scheduling*, 11:239–252, 2008.

[142] K.S. Hong and J.Y.-T. Leung. Preemptive scheduling with release times and deadlines. *Real-Time Systems*, 1:265–281, 1989. 10.1007/BF00365440.

[143] J.N. Hooker and H. Yan. A relaxation of the cumulative constraint. In *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming*, CP '02, pages 686–690, London, UK, 2002. Springer-Verlag.

[144] A. Horbach. A Boolean satisfiability approach to the Resource-Constrained Project Scheduling Problem. *Annals of Operations Research*, 181:89–107, 2010.

[145] W.A. Horn. Some simple scheduling algorithms. *Naval Research Logistics Quarterly*, 21(1):177–185, 1974.

[146] O.H. Ibarra and C.E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *J. ACM*, 22:463–468, October 1975.

[147] O. Icmeli and S.S Erenguc. A tabu search procedure for the Resource-Constrained Project Scheduling Problem with discounted cash flows. *Computers & Operations Research*, 21:841–853, October 1994.

[148] S. Irani and V.J. Leung. Scheduling with conflicts on bipartite and interval graphs. *Journal of Scheduling*, 6(3):287–307, 2003.

[149] J.R. Jackson. Scheduling a production line to minimize maximum tardiness. Technical report, University of California, Los Angeles, 1955.

[150] V. Jain and I.E. Grossmann. Algorithms for hybrid MILP/CP models for a class of optimization problems. *INFORMS Journal on Computing*, pages 258–276, 2001.

[151] K. Jansen and H. Zhang. Scheduling malleable tasks with precedence constraints. In *SPAA '05: Proceedings of the seventeenth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, pages 86–95, New York, USA, 2005. ACM.

[152] K. Jansen and H. Zhang. An approximation algorithm for scheduling malleable tasks under general precedence constraints. *ACM Transactions on Algorithms*, 2(3):416–434, 2006.

[153] J. Kallrath and J.M. Wilson. *Business Optimization Using Mathematical Programming*. MacMillan Press, 1997.

[154] K.C. Kapur. An algorithm for the project cost/duration analysis problem with quadratic and convex cost functions. *IIE Transaction*, 5:314–332, 1973.

[155] R.M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[156] J.E. Kelley Jr. Critical-path planning and scheduling: mathematical basis. *Operations Research*, 9:296–320, 1961.

[157] L.G. Khachian. A polynomial algorithm in linear programming. *Soviet Mathematics Doklady*, 20:191–194, 1979.

[158] A. Kimms. Maximizing the net present value of a project under resource constraints using a Lagrangian relaxation based heuristic with tight upper bounds. *Annals of Operations Research*, 102:221–236, 2001.

[159] R. Klein and A. Scholl. Computing lower bounds by destructive improvement: An application to Resource-Constrained Project Scheduling. *European Journal of Operational Research*, 112(2):322–346, 1999.

[160] 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.

[161] R. Kolisch and R. Padman. An integrated survey of deterministic project scheduling. *Omega*, 29(3):249–272, 2001.

[162] R. Kolisch and A. Sprecher. PSPLIB - A project scheduling problem library. *European Journal of Operational Resesearch*, 96:205–216, 1996.

[163] R. Kolisch, A. Sprecher, and A. Drexl. Characterization and generation of a general class of Resource-Constrained Project Scheduling Problems. *Management Science*, 41:1693–1703, October 1995.

[164] 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.

[165] A. Kooli, M. Haouari, L. Hidri, and E. Néron. IP-based energetic reasoning for the Resource-Constrained Project Scheduling Problem. *Electronic Notes in Discrete Mathematics*, 36:359–366, 2010. International Symposium on Combinatorial Optimization (ISCO 2010).

[166] V. Kumar. Algorithms for constraint satisfaction problems: A survey. *AI MAGAZINE*, 13(1):32–44, 1992.

[167] J. Labetoulle, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Preemptive scheduling of uniform machines subject to release dates. *Progress in Combinatorial Optimization*, pages 245–261, 1984.

[168] P. Laborie. Complete MCS-based search: Application to Resource-Constrained Project Scheduling. In *Proceedings of the 19th International Joint Conference on Artificial intelligence*, IJCAI'05, pages 181–186. Morgan Kaufmann Publishers Inc., 2005.

[169] L.R. Lamberson and R.R. Hocking. Optimum time compression in project scheduling. *Management Science*, 16(10):B597–B606, 1970.

[170] C.-Y. Lee and X. Cai. Scheduling one and two-processor tasks on two parallel processors. *IIE Transactions on Scheduling and Logistics*, 31:445–455, 1999.

[171] J.K. Lenstra, A.H.G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362, 1977.

[172] S. Leonardi, A. Marchetti-Spaccamela, and A. Vitaletti. Approximation algorithms for bandwidth and storage allocation problems under real time constraints. In *Proceedings of the 20th Conference on Foundations of Software Technology and Theoretical Computer Science*, FST TCS 2000, pages 409–420, London, UK, 2000. Springer-Verlag.

[173] R. Lepére, D. Trystram, and G.J. Woeginger. Approximation algorithms for scheduling malleable tasks under precedence constraints. *International Journal on Foundations of Computer Science*, 13(4):613–627, 2002.

[174] R. Leus and W. Herroelen. Stability and resource allocation in project planning. *IIE Transactions*, 36(7):667–682, 2004.

[175] O. Lhomme. Consistency techniques for numeric csps. In *Proceedings of the 13th International Joint Conference on Artifical Intelligence (IJCAI 1993)*, pages 232–238, 1993.

[176] K.Y. Li and R.J. Willis. An iterative scheduling technique for Resource-Constrained Project Scheduling. *European Journal of Operational Research*, 56(3):370–379, 1992.

[177] O. Liess and P. Michelon. A Constraint Programming approach for the Resource-Constrained Project Scheduling Problem. *Annals of Operations Research*, 157:25–36, 2008.

[178] S.-S. Liu and C.-J. Wang. Resource-constrained construction project scheduling model for profit maximization considering cash flow. *Automation in Construction*, 17(8):966–974, 2008.

[179] M. Lombardi and M. Milano. A precedence constraint posting approach for the rcpsp with time lags and variable durations. In I. Gent, editor, *Principles and Practice of Constraint Programming (CP 2009)*, volume 5732 of *LNCS*, pages 569–583. Springer, 2009.

[180] J.P. Marques-Silva and K.A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48:506–521, 1999.

[181] R. Marti, M. Laguna, and F. Glover. Principles of scatter search. *European Journal of Operational Research*, 169(2):359–372, March 2006.

[182] A.A. Mastor. An experimental investigation and comparative evaluation of production line balancing techniques. *Management Science*, 16(11):728–746, 1970.

[183] N. Megow, R.H. Möhring, and J. Schulz. Decision support and optimization in Shutdown and Turnaround Scheduling. *INFORMS Journal on Computing*, 23(2):189–204, 2011.

[184] L. Michel and P. Hentenryck. Activity-based search for black-box constraint programming solvers. In N. Beldiceanu, N. Jussien, and É. Pinson, editors, *Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems*, volume 7298 of *LNCS*, pages 228–243. Springer, 2012.

[185] M. Mika, G. Waligóra, and J. Weglarz. Simulated annealing and tabu search for multi-mode resource-constrained project scheduling with positive discounted cash flows and different payment models. *European Journal of Operational Research*, 164(3):639–668, 2005.

[186] A. Mingozzi, V. Maniezzo, S. Ricciardelli, and L. Bianco. An exact algorithm for the Resource-Constrained Project Scheduling Problem based on a new mathematical formulation. *Management Science*, 44(5):714–729, 1998.

[187] R.H. Möhring. Minimizing costs of resource requirements in project networks subject to a fixed completion time. *Operations Research*, 32(1):89–120, 1984.

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

[189] U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Science*, 7:95–132, 1974.

[190] R. E. Moore. *Interval analysis*. Prentice-Hall Englewood Cliffs, N.J.,, 1966.

[191] 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. ACM, 2001.

[192] D.S. Johnson M.R. Garey and A.C.-C. Yao. Resource Constrained Scheduling as Generalized Bin Packing. *Journal on Combinatorial Theory*, A(21):257–298, 1976.

[193] K. Neumann, C. Schwindt, and J. Zimmermann. *Project scheduling with time windows and scarce resources*. Springer, 2003.

[194] K. Neumann and J. Zimmermann. Procedures for resource leveling and net present value problems in projec scheduling with general temporal and resource constraints. *European Journal of Operational Research*, 127:425–443, 2000.

[195] K. Nonobe and T. Ibaraki. Formulation and tabu search algorithm for the Resource-Constrained Project Scheduling Problem. In C.C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*, pages 557–588. Kluwer Academic Publishers, 2002.

[196] W. Nuijten and C. Le Pape. Constraint-based job shop scheduling with Ilog Scheduler. *Journal of Heuristics*, 3(4):271–286, 1998.

[197] W.P.M. Nuijten and E.H.L. Aarts. Constraint satisfaction for multiple capacitated job shop scheduling. In *ECAI*, pages 635–639, 1994.

[198] CP Optimizer. http://www-01.ibm.com/software/integration/optimization/cplex-cp-optimizer/.

[199] Gurobi Optimizer. http://www.gurobi.com/.

[200] L. Özdamar and G. Ulusoy. A note on an iterative forward/backward scheduling technique with reference to a procedure by Li and Willis. *European Journal of Operational Research*, 89(2):400–407, 1996.

[201] M. Palpant, C. Artigues, and P. Michelon. LSSPER: Solving the Resource-Constrained Project Scheduling Problem with large neighbourhood search. *Annals of Operations Research*, 131:237–257, 2004.

[202] N. Pan, P. Hsaio, and K. Chen. A study of project scheduling optimization using tabu search algorithm. *Engineering Applications of Artificial Intelligence*, 21(7):1101–1112, 2008.

[203] J.H. Patterson. A comparison of exact approaches for solving the multiple constrained resource, project scheduling problem. *Management Science*, 30(7):854–867, 1984.

[204] J.H. Patterson, F.B. Talbot, R. Slowinski, and J. Weglarz. Computational experience with a backtracking algorithm for solving a general class of precedence and resource-constrained scheduling problems. *European Journal of Operational Research*, 49(1):68–79, 1990.

[205] E. Pesch and U.A.W. Tetzlaff. Constraint propagation based scheduling of job shops. *INFORMS Journal on Computing*, 8(2):144–157, 1996.

[206] C.A. Phillips, R.N. Uma, and J. Wein. Off-line admission control for general scheduling problems. In *Proceedings of the eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '00, pages 879–888. SIAM, 2000.

[207] D. Phillips and A. Garcia-Diaz. *Fundamentals of network Analysis*. Prentice-Hall, Englewood Cliffs, NJ, 1981.

[208] St. Phillips and M.I. Dessouky. Solving the project time/cost tradeoff problem using the minimal cut concept. *Management Science*, 24(4):393–400, 1977.

[209] A.A.B. Pritsker, L.J. Watters, and P.M. Wolfe. Multi project scheduling with limited resources: A zero-one programming approach. *Management Science*, 16:93–108, 1969.

[210] M. Queyranne. Structure of a simple scheduling polyhedron. *Mathematical Programming*, 58:263–285, 1993.

[211] M. Queyranne and A. Schulz. Scheduling unit jobs with compatible release dates on parallel machines with nonstationary speeds. In E. Balas and J. Clausen, editors, *Integer Programming and Combinatorial Optimization*, volume 920 of *LNCS*, pages 307–320. Springer, 1995.

[212] M. Queyranne and A.S. Schulz. Polyhedral approaches to machine scheduling. Technical report, TU Berlin, Preprint, 1994.

[213] E.T. Richards and B. Richards. Nogood learning for constraint satisfaction. In *Proceedings CP-96 Workshop on Constraint Programming Applications*. CP, 1996.

[214] A.H. Russell. Cash flows in networks. *Management Science*, 16(5):357–373, 1970.

[215] D. M. Ryan and B. A. Foster. An integer programming approach to scheduling. In A. Wren, editor, *Computer Scheduling of Public Transport Urban Passenger Vehicle and Crew Scheduling*, pages 269–280. North-Holland Publishing Company, 1981.

[216] T. Sandholm and R. Shields. Nogood learning for mixed integer programming. In *Workshop on Hybrid Methods and Branching rules in Combinatorial Optimization*, 2006.

[217] J. K. Sankaran, D. L. Bricker, and S.-H. Juang. A strong fractional cutting-plane algorithm for Resource-Constrained Project Scheduling. *International Journal of Industrial Engineering*, 6:99–111, 1999.

[218] M. Savelsbergh, Y. Wang, and L.A. Wolsey. Computational experiments with a large-scale Resource-Constrained Project Scheduling Problem. Georgia Institute of Technology, August 1996.

[219] M.W.P. Savelsbergh, R.N. Uma, and J. Wein. An experimental study of LP-based approximation algorithms for scheduling problems. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA 1998, pages 453–462. SIAM, 1998.

[220] M.W.P. Savelsbergh, R.N. Uma, and J. Wein. An experimental study of LP-based approximation algorithms for scheduling problems. *INFORMS Journal on Computing*, 17(1):123–136, 2005.

[221] M.W. Schäffter. Scheduling with forbidden sets. *Discrete Applied Mathematics*, 72(1-2):155–166, 1997. Models and Algorithms for Planning and Scheduling Problems.

[222] G. Schmidt. Scheduling with limited machine availability. *European Journal of Operational Research*, 121:1–15, 2000.

[223] L. Schrage. Solving resource-constrained network problems by implicit enumeration-preemptive case. *Operations Research*, 20(3):pp. 668–677, 1972.

[224] A.S. Schulz and M. Skutella. Scheduling-LPs bear probabilities: Randomized approximations for min-sum criteria. In *Proceedings of the 5th Annual European Symposium on Algorithms (ESA 1997)*, pages 416–429. Springer, 1997.

[225] A. Schutt. *Improving scheduling by learning.* Dissertation, The University of Melbourne, 2011.

[226] A. Schutt, G. Chu, P. Stuckey, and M. Wallace. Maximising the net present value for Resource-Constrained Project Scheduling. In N. Beldiceanu, N. Jussien, and É. Pinson, editors, *Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems*, volume 7298 of *LNCS*, pages 362–378. Springer, 2012.

[227] A. Schutt, T. Feydy, and P. Stuckey. Explaining time-table-edge-finding propagation for the cumulative resource constraint, 2012. arXiv:1208.3015v1.

[228] A. Schutt, T. Feydy, P. Stuckey, and M. Wallace. Why cumulative decomposition is not as bad as it sounds. In *Proceedings of CP 2009*, volume 5732 of *LNCS*, pages 746–761, 2009.

[229] A. Schutt, T. Feydy, P. Stuckey, and M. Wallace. Explaining the cumulative propagator. *Constraints*, 16:250–282, 2011.

[230] A. Schutt, A. Wolf, and G. Schrader. Not-first and not-last detection for cumulative scheduling in $O(n^3 \log n)$. In *Declarative Programming for Knowledge Management*, volume 4369 of *LNCS*, pages 66–80. Springer, 2006.

[231] C. Schwindt and J. Zimmermann. A steepest ascent approach to maximizing the net present value of projects. *Mathematical Methods of Operations Research*, 53:435–450, 2001.

[232] SCIP: Solving Constraint Integer Programs, 2011. http://scip.zib.de/.

[233] J. Scott. Filtering algorithms for discrete cumulative resources. Masters Thesis, 2010.

[234] T. Selle and J. Zimmermann. A bidirectional heuristic for maximizing the net present value of large-scale projects subject to limited resources. *Naval Research Logistics (NRL)*, 50(2):130–148, 2003.

[235] D. Shabtay and G. Steiner. A survey of scheduling with controllable processing times. *Discrete Applied Mathematics*, 155(13):1643–1666, 2007.

[236] N. Siemens and C. Gooding. Reducing project duration at minimum cost: A time-cost tradeoff algorithm. *OMEGA*, 3:569–581, 1975.

[237] M. Skutella. Approximation algorithms for the discrete time-cost tradeoff problem. *Mathematics of Operations Research*, 23(4):909–929, 1998.

[238] M. Skutella. *Approximization and Randomization in Scheduling.* PhD thesis, TU Berlin, Institut für Mathematik, 1998.

[239] M. Skutella. List scheduling in order of α-points on a single machine, 2002.

[240] W.E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1-2):59–66, 1956.

[241] D.E. Smith-Daniels and N.J. Aquilano. Using a late-start resource-constrained project schedule to improve project net present value. *Decision Sciences*, 18(4):617–630, 1987.

[242] D.E. Smith-Daniels and V.L. Smith-Daniels. Maximizing the net present value of a project subject to materials and capital constraints. *Journal of Operations Management*, 7(1–2):33–45, 1987.

[243] C.C. Souza and L.A. Wolsey. Scheduling projects with labour constraints. Technical Report IC-97-22, IC-UNICAMP, 1997.

[244] R.M. Stallman and G.J. Sussman. Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence*, 9(2):135–196, 1977.

[245] J.P. Stinson, E.W. Davis, and B.M. Khumawala. Multiple resource-constrained scheduling using branch and bound. *AIIE Transactions*, 10(3):252–259, 1978.

[246] F. Stork and M. Uetz. On the generation of circuits and minimal forbidden sets. *Mathematical Programming*, 102:2005, 2005.

[247] P.R. Thomas and S. Salhi. A tabu search approach for the Resource-Constrained Project Scheduling Problem. *Journal of Heuristics*, 4:123–139, 1998.

[248] E. Thorsteinsson. Branch-and-check: A hybrid framework integrating mixed integer programming and constraint logic programming. In T. Walsh, editor, *Principles and Practice of Constraint Programming (CP 2001)*, volume 2239 of *LNCS*, pages 16–30. Springer, 2001.

[249] E. Tsang. *Foundations of Constraint Satisfaction.* Academic Press, 1993.

[250] M. Uetz. *Algorithms for deterministic and stochastic scheduling.* Dissertation, TU Berlin, 2001.

[251] J.D. Ullman. NP-complete scheduling problems. *Journal of Computer and System Sciences*, 10(3):384–393, 1975.

[252] G. Ulusoy, F. Sivrikaya-Serifoglu, and S. Sahin. Four payment models for the multi-mode Resource-Constrained Project Scheduling Problem with discounted cash flows. *Annals of Operations Research*, 102:237–261, 2001.

[253] P. van Beek. Reasoning about qualitative temporal information. *Artificial Intelligence*, 58:297–326, 1992.

[254] M. Vanhoucke. http://www.projectmanagement.ugent.be/npv.html.

[255] M. Vanhoucke. New computational results for the discrete time/cost trade-off problem with time-switch constraints. *European Journal of Operational Research*, 165(2):359–374, 2005.

[256] M. Vanhoucke. A genetic algorithm for net present value maximization for resource constrained projects. In *Proceedings of the 9th European Conference on Evolutionary Computation in Combinatorial Optimization*, EvoCOP '09, pages 13–24. Springer, 2009.

[257] M. Vanhoucke. A scatter search heuristic for maximising the net present value of a resource-constrained project with fixed activity cash flows. *International Journal of Production Research*, 48(7):1983–2001, 2010.

[258] M. Vanhoucke and D. Debels. The discrete time/cost trade-off problem: extensions and heuristic procedures. *Journal of Scheduling*, 10(4–5):311–326, 2007.

[259] M. Vanhoucke, E. Demeulemeester, and W. Herroelen. On maximizing the net present value of a project under renewable resource constraints. *Management Science*, 47:1113–1121, August 2001.

[260] M. Vanhoucke, E. Demeulemeester, and W. Herroelen. Discrete time/cost trade-offs in project scheduling with time-switch constraints. *Journal of Operational Research Society*, 53:1–11, 2002.

[261] P. Vilím. Edge finding filtering algorithm for discrete cumulative resources in $O(kn \log n)$. In *Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming, CP 2009*, pages 802–816. Springer, 2009.

[262] P. Vilím. Max energy filtering algorithm for discrete cumulative resources. In W.-J. van Hoeve and J. Hooker, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 5547 of *LNCS*, pages 294–308. Springer, 2009.

[263] P. Vilím. Timetable edge finding filtering algorithm for discrete cumulative resources. In T. Achterberg and C. Beck, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 6697 of *LNCS*, pages 230–245. Springer, 2011.

[264] H. Wang, T. Li, and D. Lin. Efficient genetic algorithm for Resource-Constrained Project Scheduling Problem. *Transactions of Tianjin University*, 16:376–382, 2010.

[265] F. Yalaoui and C. Chu. New exact method to solve the $Pm|r_j|\sum C_j$ schedule problem. *International Journal of Production Economics*, 100(1):168–179, 2006.

[266] H.-H. Yang and Y.-L. Chen. Finding the critical path in an activity network with time-switch constraints. *European Journal of Operational Research*, 120(3):603–613, 2000.

[267] K.K. Yang, F.B. Talbot, and J.H. Patterson. Scheduling a project to maximize its net present value: An integer programming approach. *European Journal of Operational Research*, 64(2):188–198, 1993.

[268] R. Zabih and D. McAllester. A rearrangement search strategy for determining propositional satisfiability. In *Proceedings of the National Conference on Artificial Intelligence*, pages 155–160, 1988.

[269] J. Zhan. Calendarization of time planning in MPM networks. *ZOR – Methods and Models for Operations Research*, 36(5):423–438, 1992.

[270] 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. IEEE Press, 2001.

[271] D. Zhu and R. Padman. A metaheuristic scheduling procedure for resource-constrained projects with cash flows. *Naval Research Logistics (NRL)*, 46(8):912–927, 1999.

[272] David Zuckerman. On unapproximable versions of NP-complete problems. *SIAM Journal on Computing*, 25(6):1293–1304, 1996.