Presolving techniques and linear relaxations for cumulative scheduling

vorgelegt von
Dipl.-Math. oec. Stefan Heinz
geb. in 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. Peter Bank
Gutachter:	Prof. Dr. J. Christopher Beck
Gutachter:	Prof. Dr. Thorsten Koch

Tag der wissenschaftlichen Aussprache: 7. Mai 2018

Berlin 2018

Abstract

In the mathematical optimization community the term *scheduling* usually describes the computation of a sequential plan for a set of jobs w.r.t. a set of side conditions such as precedence constraints and resource restrictions. Thereby, a certain objective should be fulfilled which can be for example to minimize the latest completion time of all jobs. The sequential plan can be an ordering of the jobs or in case of this dissertation a schedule which assigns a start time to each job.

Many scheduling problems can be modeled and solved as a constraint program as well as a mixed-integer (linear) program. Both approaches have their advantages and disadvantages which are often complementary. In this dissertation we use a hybrid approach, called constraint integer programming, to solve scheduling problems. We focus on scheduling problems which contain a cumulative resource structure: the available resources are renewable and can be shared between jobs which have to be scheduled non-preemptively.

We define the class of energy-based propagation algorithms which are inference algorithms for the cumulative resource structure using volume arguments to infer bounds on the start time and the completion time of a job. Many of the known propagation algorithms for the cumulative resource structure, such as time-tabling, edge-finding, time-tabling edge-finding, and energetic reasoning, belong to this class. For this class we develop explanations for their inferred bound changes. These explanations are used during the analyzes of infeasible sub-problem to retrieve additional (redundant) constraints which help to solve a particular problem more quickly. This concept generalizes known explanations for propagation algorithms for the cumulative resource structure. In addition we show that each energy-based propagation algorithm implies a linear relaxation for the cumulative resource structure with optional jobs.

For current state-of-the-art mixed-integer programming solvers presolving is an important feature. During presolving, the original problem is reformulated into a hopefully easier-to-solve problem. One aim is to remove redundant variables and constraints. For the cumulative resource structure we present several presolving techniques generalizing the concept of dual reductions, which is known for mixed-integer programs to shrink a problem formulation, to constraint programs. This techniques allows us to remove feasible or even optimal solutions from the solution space as long as one optimal solution remains in case that the problem is feasible. Using this idea we develop several dual reduction steps for the cumulative resource structure. These reductions enable the removal of jobs from a cumulative resource with the knowledge that this job can be scheduled independently of the schedule for the remaining jobs.

In a computational study, we analyze the impact of the presolving techniques for the cumulative constraint and the linear relaxations for the cumulative constraint with optional jobs. Therefore, we use two problem classes which are resource-constrained project scheduling problems and resource allocation and scheduling problem.

Zusammenfassung

In der mathematischen Optimierung bezeichnet der Begriff Scheduling die Berechnung eines Ablaufplans, die eine gegebenen Menge von Prozessen (Jobs), typischerweise mit Reihenfolgebeziehungen, einer Menge von beschränkten Ressourcen (z.B. Maschinen) zuordnet, meist mit dem Ziel die späteste Fertigstellungszeit aller Jobs zu minimieren.

Scheduling-Probleme lassen sich sowohl als Gemischt-Ganzzahlige (Lineare) Programme als auch als Constraint-Programme modellieren und lösen. Beide Ansätze haben ihre Stärken und Schwächen, diese sind aber in vielerlei Hinsicht komplementär. In der vorliegenden Arbeit verwenden wir einen integrierten Ansatz, die sogenannte Constraint-Ganzzahlige Programmierung, um Scheduling-Probleme zu lösen. Wir konzentrieren uns auf Scheduling-Probleme mit Kumulativ-Bedingungen, d.h. es gibt Ressourcen, welche sich mehrere nicht-unterbrechbare Jobs teilen.

Wir definieren die Klasse von Energie-basierten Propagierungsalgorithmen für Kumulativ-Bedingungen, welche Volumenargumente nutzen, um die Startzeit und Endzeit eines Jobs zu beschränken. Viele aus der Literatur bekannte Propagierungsalgorithmen, wie time-tabling, edge-finding, time-tabling edge-finding und energetic reasoning, gehören zu dieser Klasse. Für die Klasse der Energie-basierten Propagierungsalgorithmen entwickeln wir allgemeine Erklärungen für die propagierten Schranken. Diese Erklärungen werden in der Analyse von unzulässigen Teilproblemen genutzt, um zusätzliche gültige Bedingungen zu lernen. Dabei generalisieren wir aus der Literatur bekannte Erklärungen für Propagierungsalgorithmen für Kumulativ-Bedingungen. Zusätzlich zeigen wir, dass jeder Energie-basierte Propagierungsalgorithmus eine lineare Relaxierung für Kumulativ-Bedingung mit optionalen Jobs impliziert. Diese Relaxierung kann genutzt werden, um die lineare Relaxierung eines Constraint-Ganzzahligen Programms zu verstärken.

Presolving-Verfahren sind ein wichtiger algorithmischer Bestandteil zum Lösen Gemischt-Ganzzahlige Programme. Presolving-Verfahren reformulieren das Original-Problem in ein (hoffentlich) leichter zu lösendes Problem. Oftmals werden dabei redundante Variablen und Bedingungen eliminiert. Wir entwickeln diverse Presolving-Verfahren für Kumulativ-Bedingungen vor. Unter anderem verallgemeinern wir das Konzept der dualen Reduktion, welches in der Gemischt-Ganzzahlige Programmierung für Problemvereinfachungen genutzt wird, auf die Constraint-Programmierung. Solch eine Reduktion kann genutzt werden, um zulässige oder sogar optimale Lösungen aus dem Lösungsraum zu entfernen, solange sicher gestellt wird, dass nicht alle Optimallösung entfernt werden. Wenn es beispielsweise eine Bearbeitungszeit für einen Job gibt, aus der sich keinerlei Einschränkungen für andere Jobs ergeben, kann dieser dort platziert werden und aus der entsprechenden Kumulativ-Bedingung entfernt werden.

In einem umfangreichen Rechenexperiment zeigen wir, dass der im Rahmen dieser Arbeit entwickelte Scheduling-Löser kompetitiv zu den besten aus der Literatur bekannten Scheduling-Lösern ist. Dabei ist zu bemerken, dass die entwickelten Techniken allgemein gültig sind für die Constraint-Ganzzahlige Programmierung. Das heißt, diese Techniken können allgemein zum Lösen aller Gemischt-Ganzzahligen Programme, die Kumulativ-Bedingungen enthalten, genutzt werden und sind nicht problemspezifisch.

Acknowledgment

Thanks all for your patience!

Contents

Introduction

Optimization problems can be addressed by diverse modeling and solving techniques that have been developed in different areas of Mathematics and Computer Science research. Common approaches include constraint programming [RBW06] and mixed-integer programming [NW88]. For either of them, there exist applications for which they appear as the natural method of choice. On the one hand, constraint programming typically excels for highly combinatorial problems, for problems with a nonlinear, discrete structure, and for pure feasibility problems. Examples include combinatorial puzzles and verification problems. Mixed-integer programming shines for problems which include continuous variables, for problems which have a strong linear relaxation, and for problems for the hardness consists of proving optimality. Examples include unit commitment and network flow problems.

There are, however, applications which are equally challenging for both constraint programming and mixed-integer programming. A prime example is scheduling. It is an active field of research in both communities and a canonical candidate for the application of hybrid approaches [Hoo11, MH11]. Scheduling problems arise with many different characteristics in practice [BPN01, Bru01]. In this dissertation, we concentrate on the resource-constrained project scheduling problem [BDM$^+$99] and on the resource allocation and scheduling problem [Hoo04, Hoo05a]. Both can be modeled using the so-called *cumulative* constraint [AB93], one of the classical global constraints [BCDP07] in constraint programming. It is used to describe a relationship between a renewable resource, e.g., a machine, and non-preemptive jobs which require a certain amount of the resource during their execution.

The dissertation studies algorithms for presolving, propagation, and conflict analysis using cumulative constraints and evaluates their effectiveness within an integrated constraint and integer programming framework.

This dissertation is subdivided into five chapters, one for introducing basic notations and concepts, two mostly theoretic chapters which propose new algorithms and extensions of existing algorithms to efficiently deal with cumulative constraints, and two mainly computational chapters which present numerical results for the resource-constrained project scheduling problem and for the resource allocation and scheduling problem. It concludes with a summary and outlook chapter.

Outline and contributions of the dissertation

In Chapter 1 we introduce the basic notations and concepts used throughout the dissertation. This includes a definition of constraint programming, mixed-integer programming, and related problem classes as well a solving approach for these types of problem. Furthermore, we discuss the cumulative constraint, which plays a central role in the work, in detail. Besides a formal definition we recall those propagation algorithms for this global constraint which are of interest in this dissertation.

In Chapter 2, we consider propagation algorithms for the cumulative constraints and relaxations derived therefrom. We introduce a generalized class of propagation algorithms, the *energy-based propagation algorithms* and describe how existing propagation algorithms relate to this class. We develop explanations for their inferences, generalizing previously existing results. The conflict constraints derived from those explanations are used to create the *conflict relaxation*, which we formally define. This relaxation is dynamically tightened during the search by the analysis of infeasible sub-problems.

Furthermore, we present new linear relaxations for the cumulative constraint with optional jobs in this chapter. We prove that each propagation algorithm belonging to the class of energy-based propagation algorithm implies a linear relaxation.

In Chapter 3, we develop various presolving algorithms for single cumulative constraints as well as for sets of cumulative constraints. Presolving algorithms for scheduling typically need a precedence graph of the jobs as an input. We introduce an algorithm which projects a variable bound graph of an arbitrary optimization problem to a precedence graph. We exploit this to extend problem-specific presolving techniques to general optimization problems with cumulative constraints. We also discuss the detection of disjunctive constraints in order to apply stronger domain propagation.

Furthermore, we generalize the concept of dual reductions – which is well known for mixed-integer programming – to constraint programming. We show that for classical scheduling benchmarking instances, this leads to a reduction of the problem size. Finally, the developed presolving algorithms are extended to the cumulative constraints with optional jobs.

In Chapter 4, we present the first of two comprehensive computational studies. We consider *resource-constrained project scheduling problems*. The goal of resource-constrained project scheduling problems is to schedule jobs on renewable resources subject to precedence constraints while minimizing the makespan. We evaluate the impact of different propagation algorithms for cumulative constraints. We analyze the effect of applying primal and dual presolving techniques. We show results which compare the general solver SCIP, including our implementation of a cumulative constraint handler, against a state-of-the-art solver for resource-constrained project scheduling problems.

In Chapter 5, we consider scheduling problems with optional jobs, more pecisely, resource allocation and scheduling problems. That means, that in addition to the ordering of jobs, an assignment of each job to a particular resource has to be made. We utilize the linear relaxation which we introduced for the cumulative constraint with optional jobs and present a logic-based Benders approach and a mixed-integer programming approach. We compare them against a state-of-the-art logic-based Benders implementation and classify the individual strenghts of each of the approaches.

All algorithms that are presented within this dissertation are implemented within SCIP. They are freely available in source code for all academic users at `http://scip.zib.de`. Partial work and preliminary results of this dissertation have been published in [BHL$^+$10, HS11, BHS11, HB11, HB12a, HKB13, HSB13].

1 Constraint optimization problems

Constraint optimization problems (COP) are a very rich problem class that has almost no restriction w.r.t. the side constraints and objective function. Many scheduling problems fit into that paradigm. *Mixed-integer programming (MIP)* and *satisfiability testing (SAT)* are special cases of the general idea of COP. The power of COP arises from the possibility to model a given problem with a large variety of expressive constraints [RBW06]. In contrast, SAT and MIP only allow for very specific constraints: Boolean clauses for SAT and linear and integrality constraints for MIP. Their advantage, however, lies in the sophisticated techniques available to exploit the structure provided by these restrictive constraint types.

The goal of *constraint integer programming (CIP)* is to combine the advantages and compensate for the weaknesses of CP, MIP, and SAT. It was introduced by Achterberg [Ach04, Ach07b, Ach09] and implemented in the framework SCIP [Ach09, SCI].

The central solving approach for CIP as implemented in the SCIP framework is branch-and-cut-and-propagate: as in SAT, CP, and MIP-solvers, SCIP performs a branch-and-bound search to decompose the problem into sub-problems. As in MIP, a linear relaxation, potentially strengthened by additional inequalities/cutting planes, is solved at each search node and used to guide and bound the search. Similar to CP solvers, inference in the form of constraint propagation is used at each node to further restrict search and detect dead-ends. Moreover, as in SAT solving, SCIP uses conflict analysis to learn from infeasible sub-problems and restarts.

Outline. This chapter is organized as follows. In Section 1.1 we define the different problem classes COP, CIP, MIP, and SAT formally. A solving approach for COPs is recalled in Section 1.2. In Section 1.3 we focus on special structures which are of interest for this work. The cumulative constraint is introduced in Section 1.4.

1.1 Problem definition

In this section we recall a definition for a constraint optimization program (COP) and introduce special problem classes which are of interest for this dissertation.

Definition 1.1 (constraint optimization problem). A *constraint optimization problem* $\text{COP} = (X, \mathfrak{C}, \mathfrak{D}, f)$ consists in solving

$$\text{(COP)} \quad c^\star = \min\{f(X) : \mathfrak{C}(X) = 1,\ X \in \mathfrak{D}\}$$

with $\mathfrak{D} = \mathcal{D}_1 \times \cdots \times \mathcal{D}_n$ representing the domains of finitely many variables $X = (x_1, \ldots, x_n)$, with $n \in \mathbb{N}$, a finite set $\mathfrak{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_m\}$ of constraints $\mathcal{C}_i : \mathfrak{D} \to \{0, 1\}$, $i = 1, \ldots, m$ with $m \in \mathbb{N}$, and an objective function $f : \mathfrak{D} \to \mathbb{R}$.

We remark that for a given variable assignment, that is $\boldsymbol{x} \in \mathfrak{D}$ with $x_j \in \mathcal{D}_j$ for $j = 1, \ldots, n$, a constraint $\mathcal{C}$ indicates whether it is feasible (one) or violated (zero). We restrict ourselves w.l.o.g. to minimization problems to keep the notation clear.

Restriction 1.2 (integer domains). For a COP, domains can be, for example, discrete, continuous, or even power sets. In this dissertation we restrict ourselves to variable domains being integers and finite. If results hold for more general domains, we state that explicitly.

COPs are a very general class of problems. Special classes which are of insterest in this work are satisfiability problems (SAT), mixed-integer programs (MIP), and constraint integer programs (CIP). In the following we define these problem classes.

Definition 1.3 (satisfiability problem). A *satisfiability problem* (SAT) is a COP $= (X, \mathfrak{C}, \mathfrak{D}, f)$ where the objective function f is the constant zero, the variable domains are restricted to Booleans, that is $\mathfrak{D} = \{0,1\}^n$, and the constraints are restricted to clauses. A clause is a disjunction of literals $\mathcal{C}_i = \ell_1^i \vee \ldots \vee \ell_{k_i}^i$. A literal $\ell \in \{x_1, \ldots, x_n, \bar{x}_1, \ldots, \bar{x}_n\}$ is either a variable x_j or the negation of a variable $\bar{x}_j = (1 - x_j)$. A clause is satisfied if at least one literal is true.

As the name already suggests, these problems are satisfiability problem asking if there exists an assignment to the Boolean variables such that each clause is satisfied. For more details on this problem class we refer to the “Handbook of Satisfiability” [BHvMW09].

Definition 1.4 (mixed-integer program). A mixed-integer program (MIP) is a COP $= (X, \mathfrak{C}, \mathfrak{D}, f)$ where the objective function is linear, the variable domains are restricted to real or integer intervals, and the constraints are linear inequalities.

If all integer domains are relaxed to be continuous, we have the *linear programming (LP) relaxation* of a MIP. An LP can be solved efficiently in practice using the simplex method [Dan51] or a barrier approach, see, e.g., [NN94]. A solution to this relaxation satisfies all (linear) constraints, but might violate the integrality conditions for the variable domains. It provides a lower bound on the optimal objective function.

The concept of an LP relaxation is used within LP-based branch-and-bound [Dak65], the method by which MIPs are typically solved. In the last decades, MIP solvers improved tremendously w.r.t. speed and the complexity of instances which these solvers can handle [BFG$^+$00, AW13]. For more details on mixed-integer programming, we refer to [NW88].

Definition 1.5 (constraint integer program). A *constraint integer program* CIP is a COP $= (X, \mathfrak{C}, \mathfrak{D}, f)$ for which the objective function is linear, the variable domains are restricted to real and integer intervals, and the constraint set satisfies the restriction that after all variables with integer domains are fixed the remaining problem is a linear program.

For more details on constraint integer programming we refer to [Ach07b].

Figure 1.1 visualizes the inclusion of the different problem classes. It shows that SAT is subclass of MIP which is again a subclass of CIP. COP is a superset of all the three special problem classes.

For a COP, we define the set of feasible and optimal solutions with X_{COP} and $X^\star_{\text{COP}}$, respectively. That is

$$X_{\text{COP}} := \{\boldsymbol{x} \,:\, \boldsymbol{x} \in \mathfrak{D}, \mathfrak{C}(\boldsymbol{x}) = 1\}, \qquad X^\star_{\text{COP}} := \{\boldsymbol{x} \,:\, f(\boldsymbol{x}) = c^\star, \boldsymbol{x} \in X_{\text{COP}}\}$$

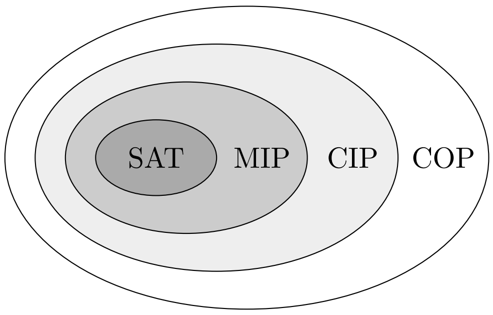


Figure 1.1: Visualizing of the inclusions of the different problem spaces: constraint optimization problems (COP), constraint integer program (CIP), mixed-integer programs (MIP), and satisfiability testing (SAT).

where $c^\star$ is the minimum objective function value of any feasible solution (see Definition 1.1).

1.2 Solving techniques

In this dissertation, we restrict ourselves to COPs where the variable have finite integer domains (see Restriction 1.2). Such COPs can be solved by enumerating all possible assignments for variables. To avoid explicitly testing all assignments, a systematic search can be performed combined with sophisticated techniques which remove infeasible and sub-optimal assignments early.

One way of doing this is to recursively split the problem into smaller sub-problems, thereby creating a search tree. In each search node a variety of techniques are used to remove assignments which are infeasible or sub-optimal. This approach is known as branch-and-bound [LD60] and is widely used in commercial and academic solvers to solve sub-classes of COPs.

In this section we briefly sketch the techniques which are of interest for this work. These are domain reductions and the use of a linear relaxation. The analysis of infeasible sub-problems is addressed separately in Chapter 2.

1.2.1 Domain reductions

Each sub-problem contains restrictions which are added during the splitting of the parent sub-problem into smaller sub-problems. These restrictions can be used to infer further reductions, thereby shrinking the feasibility region.

Propagation algorithms are used to shrink variable domains. Such an algorithm takes the variable domains and a subset of constraints as input and returns a smaller or same size domain space. Usually, it is guaranteed that it does not remove any feasible solution. Such a reduction is called *primal feasible.*

Definition 1.6 (primal feasible). Given a COP $= (X, \mathfrak{C}, \mathfrak{D}, f)$. For any variable x_j, a domain reduction $\mathcal{D}'_j \subset \mathcal{D}_j$ is called *primal feasible* if for all $\boldsymbol{x} \in X_{\text{COP}}$ it holds that $x_j \in \mathcal{D}'_j$.

Since we are focusing on constraint optimization problems, where the goal is to find an optimal solution, we can allow propagation algorithms to remove feasible or even optimal solutions as long as it is guaranteed that at least one optimal solution remains. Such a domain reduction is called *dual feasible*.

Definition 1.7 (dual feasible). Given a COP $= (X, \mathfrak{C}, \mathfrak{D}, f)$. For any variable x_j, a domain reduction $\mathcal{D}'_j \subset \mathcal{D}_j$ is called *dual feasible* if the non-reduced COP is infeasible or there exists an optimal solution $\boldsymbol{x} \in X^{\star}_{\text{COP}}$ where $x_j \in \mathcal{D}'_j$.

Remark 1.8. Note that a dual feasible domain reduction is an extension of the unifying framework for structural properties of constraint satisfaction problems introduced by Bordeaux et al. [BCM08]. Their conditions have to hold statically for each feasible (optimal) solution. This is *not* the case for a dual feasible reduction. We only assume that after applying a domain reduction there still exists a feasible (optimal) solution for the original problem if the problem is feasible at all. There is no restrictions on the assignments of the other variables.

A special form of a domain reduction is a bound change. Since we consider integer domains, a bound change can be defined as follows.

Definition 1.9 (bound change). We distinguish for a variable x_i between lower and upper bound change. A *lower bound change* for x_i is a hyperplane of the form

$$[\![x_i \geq a]\!] = \{\boldsymbol{x} : x_i \geq a\}$$

whereas an *upper bound change* is

$$[\![x_i \leq b]\!] = \{\boldsymbol{x} : x_i \leq b\}.$$

1.2.2 Linear relaxation

For MIPs, the linear programming relaxation omits the integrality conditions. An optimal solution of the LP relaxation satisfies all linear constraints and provides a lower bound for the optimal solution value of the MIP. If in addition all integrality conditions are respected the LP solution provides an optimal solution for the corresponding MIP. If, however, the LP solution contains fractional solution values for variables which have an integer domain in the MIP, then these variables are "good" candidates to split the problem. After selecting one of these variables, the solution value of the variable in the LP solution defines how the variable domain can be split. Therefore, the LP relaxation of a MIP can be used to drive the search by providing candidates for branching.

SAT problems are a special case of MIPs. For SAT problem, where we have no objective function, assigning to all variables $\frac{1}{2}$ is feasible[1] for the corresponding LP relaxation. That means all variables might have a fractional solution value in a solution of the LP relaxation. In that case the LP relaxation does not restrict the candidates for branching and is not useful to guide the search.

For general COPs, we have the same issue w.r.t. search guidance, however, due to a different reason. Usually a COP has no unique LP relaxation because the individual constraints of a COP often have no unique linear relaxation. For example, the cumulative

[1]Ignoring fixed variables and assuming each clause contains at least 2 literals.

constraint with optional jobs (see Section 1.4.4) has different linear relaxation (see Section 2.3). Hence, the LP relaxation of a COP depends on the chosen linear relaxation for each constraint. In addition, if a solution of an linear programming relaxation satisfies all integrality conditions, the solution might still not be feasible for the COP. Since the LP is a relaxation, it still provides a lower bound but the search guidance is not as strong as for a MIP. For example if all variables with integer domain have integral values in an LP relaxation, the LP relaxation does not directly provide any candidate as branch variable. Furthermore, it is not clear how the variable domain should be split. As a consequence it is not clear if variables with fractional solution value in an LP relaxation are "good" candidates for branching. Often, however, the LP relaxation provides enough information to select a branching.

1.3 Sources of globally valid structures

In the previous section we discussed how COP with integer variable domain can be solved. This section is dedicated to special problem structures which can be detected and used to improve the solving process of COPs. For MIP solvers, which are restricted to linear constraints and a linear objective, there are few problem specific structures that can be passed the the solve. MIP solvers, however, try to detect useful structures to take advantage of them during the solving process. In contrast, CP solvers follow a different strategy. In CP, the modeler communicates structure via global constraints and can even create a new global constraint for structures which they want to handle [BCDP07].

The structures which are of interest in this dissertation are the variable bound graph and the variable locks. In the following we introduce these structures in more detail.

1.3.1 Variable bound graph

A *variable bound constraint* between two variables x and y has one of the following forms

$$b \cdot x + c \leq y \quad \text{or} \quad b \cdot x + c \geq y \tag{1.1}$$

with $b \in \mathbb{R} \setminus \{0\}$ and $c \in \mathbb{R}$. The coefficients b and c are called the *variable bound coefficient* and the *variable bound constant*, respectively. The first inequality is a variable lower bound constraint since x bounds the lower bound of y. The second inequality is a variable upper bound constraint. Here x bounds y from above. Depending on the variable bound coefficient b, variable y also bounds the lower or upper bound of variable x. Consequently, a variable bound relation expresses the dependency of one bound of a variable on a bound of another variable. Typical examples for the use of variable bound constraints are precedence constraints on start time variables in scheduling (see Section 3.1) or big-M constraints modeling fixed-costs in production planning [DDK12].

Variable bound relations cannot only be deduced from variable bound constraints, but can also be identified within more general constraints (see Section 3.3.3) or during presolving, e.g., by probing [Sav94]. These relations are exploited by different solver components, e.g., for c-MIR cut separation, where they can be used to replace non-binary variables with binary variables [MW01], lifting flow cover inequalities [GNS96], or primal heuristics which take advantage of the variable bound graph to construct a neighborhood containing hopefully "good" feasible solutions [GBHW15]. In this work we present algorithms which use variable bound constraints together with a cumulative constraint to strengthen other variable bound constraints (see Section 3.3.4).

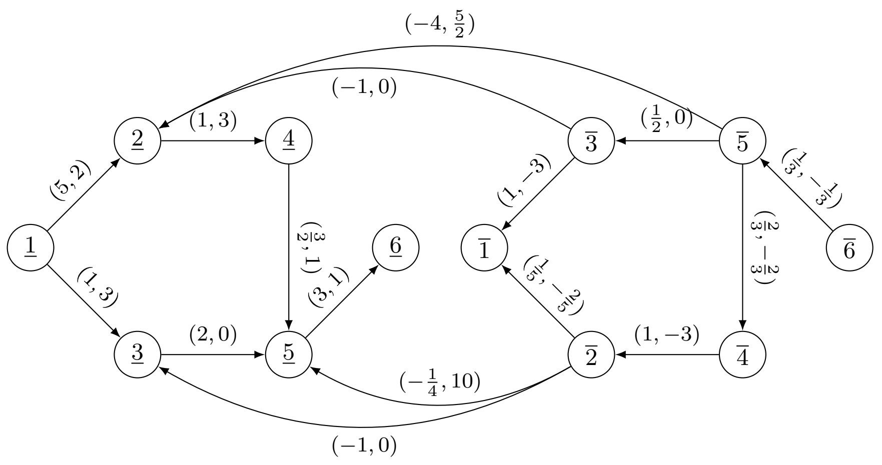


Figure 1.2: The figure displays a variable bound graph for the variable bound constraints given in Example 1.10. A vertex corresponds to one of the bounds of a variable. The vertex number refers to the variable name and the under (over) line indicates if it is the lower (upper) bound of the variable. Each arc is labelled with a tuple (b, c) stating the variable bound coefficient b and the variable bound constant c.

In order to exploit variable bound relations, we stored them in a global structure. They form the *variable bound graph*, a directed graph in which each node corresponds to a lower or upper bound of a variable. Each variable bound relation is then represented by an arc in the graph, which points from the influencing bound to the dependent bound. That implies that each variable bound constraint corresponds to two arcs in the variable bound graph. In addition, each arc is equipped with the variable bound coefficient and the variable bound constant. Hence, the global structure of variable bounds consists of a directed graph $D = (V, A)$ with variable bound coefficient mapping $b : A \to \mathbb{R}$ and variable bound constant mapping $c : A \to \mathbb{R}$. A vertex of the graph corresponds to a lower or upper bound of a variable. Hence, the number of vertices is limited by two times the number of variables. The following example illustrates the variable bound graph.

Example 1.10. Given six variables $x_1, \dots, x_6$ and the following variable bound constraints:

$$\begin{array}{rcr} x_1 - x_3 \le -3 & 5x_1 - x_2 \le -2 & -x_2 - x_3 \le 0 \\ 2x_3 - x_5 \le 0 & -0.25x_2 - x_5 \le -10 & x_2 - x_4 \le -3 \\ -0.5x_4 - x_4 \le -8 & 1.5x_4 - x_5 \le -1 & 3x_5 - x_6 \le -1 \end{array}$$

Figure 1.2 depicts the variable bound graph. For each variable bound constraint two arcs are added to the graph.

This graph can be used to read all implications of bound change by following all paths starting from the corresponding vertex.

1.3.2 Variable locks

Achterberg [Ach07b] defined a mechanism, called *variable locks*, to implement dual reductions based on global information for CIP problems. Essentially, a variable lock represents information about the relationship between a variable and a set of constraints. Achterberg used this information during presolving to infer dual reductions for mixed-integer linear programs within a constraint-based system. Building on the existing idea of variable locks, we formally define and justify the use of dual information for constraint optimization problems.

In this section, we generalize variable locks to constraint optimization programs and show how these locks can be used to infer dual reductions. As our results hold for both constraint satisfaction problems and constraint optimization problems, we introduce them in the more general optimization context.

While domains can be, for example, discrete, continuous, or even power sets, variable locks rely on the variable domains being totally ordered w.r.t. to the relation "$\leq$". The relation "$\leq$" is a total order on a set M if, for all $a, b, c \in M$, it is true that: (i) if $a \leq b$ and $b \leq a$ then $a = b$ (antisymmetry); (ii) if $a \leq b$ and $b \leq c$ then $a \leq c$ (transitivity); and (iii) $a \leq b$ or $b \leq a$ (totality).

The basic idea of the variable locks is to maintain a count, for each variable, of the number of constraints that might become violated by increasing or decreasing the value of the variable. To define the variable locks formally, we define the property that a constraint is *monotone decreasing or increasing in a variable.*

Definition 1.11. A constraint $\mathcal{C} : \mathfrak{D} \to \{0, 1\}$, is *monotone decreasing (increasing) in variable* x_j, if for all assignments $\dot{x} \in \mathfrak{D}$ which are feasible for constraint $\mathcal{C}$, that is $\mathcal{C}(\dot{x}) = 1$, it holds that for all assignments $\hat{x} \in \mathfrak{D}$ with $\dot{x}_k = \hat{x}_k$ for all $k \neq j$ and $\hat{x}_j < \dot{x}_j$ ($\hat{x}_j > \dot{x}_j$) are also feasible for constraint $\mathcal{C}$, that is $\mathcal{C}(\hat{x}) = 1$.

If a constraint is either monotone decreasing or increasing in each variable in its scope, it is a *monotone constraint* (see Dechter [Dec03]). Depending on the monotone status of a constraint, variable locks can be omitted.

Definition 1.12. Given a constraint $\mathcal{C} : \mathfrak{D} \to \{0, 1\}$. The constraint $\mathcal{C}$ needs to *down-lock* (*up-lock*) variable x_j if and only if the constraint is not monotone decreasing (increasing) in variable x_j. That is, if and only if there exist two vectors $\dot{x}, \hat{x} \in \mathfrak{D}$ with $\mathcal{C}(\dot{x}) = 0$, $\mathcal{C}(\hat{x}) = 1$, $\dot{x}_k = \hat{x}_k$ for all $k \neq j$, and $\dot{x}_j < \hat{x}_j$ ($\dot{x}_j > \hat{x}_j$).

Given a variable x_j with a totally ordered domain, a constraint $\mathcal{C}$ does not need to down-lock (up-lock) x_j if, for any feasible assignment $\hat{x}$, *any* assignment $\dot{x}$ that differs from $\hat{x}$ only in that the value of x_j is smaller (greater) is also feasible. This definition directly yields the following corollary.

Corollary 1.13. Given a constraint $\mathcal{C} : \mathfrak{D} \to \{0, 1\}$. A variable x can be removed from the scope of constraint $\mathcal{C}$ if this constraint is monotone decreasing and increasing in variable x (i.e., it does not lock variable x in any direction).

To be able to remove such a "completely free" variable from a constraint, some adjustment to the particular constraint may be necessary (see Lemma 3.10 as example).

Individual locks can be aggregated into dual information for a set of constraints. Here, following Achterberg [Ach07b], we accumulate locks by simply counting the number of constraints that down- or up-lock a variable, respectively. For a given constraint optimization problem, the *(accumulated) variable locks*, ζ_j^- and ζ_j^+, can be interpreted as the number of constraints that "block" the shifting of x_j towards its lower or upper bound.

Example 1.14. Given four integer variables $x_1, x_2, x_3, x_4 \in \{0, \dots, 10\}$ and the following linear constraint system:

$$\begin{aligned} 5\,x_1 - 6\,x_2 + x_4 &\le 8 \\ x_1 + x_3 &= 1 \end{aligned}$$

The locks are: $\zeta_1^+ = 2$, $\zeta_1^- = 1$, $\zeta_2^+ = 0$, $\zeta_2^- = 1$, $\zeta_3^+ = 1$, $\zeta_3^- = 1$, $\zeta_4^+ = 1$, and $\zeta_4^- = 0$.

Remark 1.15. For linear constraints the variable locks are predefined and independent of the variable domains. This means, they do not change if the variable domains are tightened. This is not the case for example for cumulative constraints (see Section 3.3.1).

A MIP solver usually has access to the column representation of the problem matrix. That allows it to efficiently identify in which constraints each variable appears with a non-zero coefficient. That knowledge allows a solver to perform dual reductions in a sound way. Variable locks are not as informative as the column representation and, in fact, can be seen as a relaxation. However, a substantial number of dual reductions performed in a MIP solver can be done using only the variable locks [Ach07b].

Consider a variable x_j with a totally ordered domain and no down-locks ($\zeta_j^- = 0$), that is, all constraints are monotone decreasing in variable x_j. If there exists a feasible solution $\hat{x}$ with $\hat{x}_j \neq \min\{d \in \mathcal{D}_j\}$, then it follows that the solutions $\dot{x}$ with $\dot{x}_k = \hat{x}_k$ for all $k \neq j$ and $\dot{x}_j \in \{d \in \mathcal{D}_j \,:\, d < \hat{x}_j\}$ are also feasible. Therefore, fixing this variable to its lower bound is a valid inference w.r.t. the feasibility of the problem. This is the case for variable x_4 in the above example. In an optimization context, such a fixing can only be performed if the objective function, which we assume is to be minimized, is monotonically non-decreasing[2] in this variable. A symmetric argument holds for up-locks. Hence, each variable that has a down-lock (up-lock) of zero and the objective function is monotonically non-decreasing (non-increasing) in this variable can be fixed to its lower (upper) bound.[3] Such an inference is dual feasible and is called a *dual fixing*.

As noted, using variable locks to detect such "half free" variables was already presented [Ach07b]. The following lemma summarizes this idea of dual fixing.

Lemma 1.16. Given a COP $= (X, \mathfrak{C}, \mathfrak{D}, f)$. If a variable x_j with totally ordered domain $\mathcal{D}_j$ has $\zeta_j^- = 0$ ($\zeta_j^+ = 0$) and the objective function is monotonically non-decreasing (non-increasing) in x_j, then fixing this variable to $x_j = \min\{d \in \mathcal{D}_j\}$ ($x_j = \max\{d \in \mathcal{D}_j\}$) is dual feasible.

Example 1.17. Reconsider the linear constraints from Example 1.14. Given, additionally, an objective function $f(\boldsymbol{x}) = x_1 + x_2 + x_3 + x_4$ to be minimized, the variable x_4 can be dual fixed to its lower bound. In contrast, variable x_2 cannot be fixed to its upper bound since the objective function is not monotone non-increasing in x_2.

[2]A function $f(x)$ is called to be monotonically non-decreasing (non-increasing) on an interval I if $f(b) \geq f(a)$ ($f(b) \leq f(a)$) for all $b > a$ where $a, b \in I$.

[3]For a variable that is not directly involved in the objective function, the objective function is both monotonically non-decreasing and non-increasing w.r.t. that variable.

In practice, the accumulated variable locks can be an overestimate of the actual variable locks since each constraint can guarantee completeness by just locking its variables in both directions without considering Definition 1.12. Such an overestimate is a relaxation and is still usable. However, if a constraint does not lock a variable where it should w.r.t. Definition 1.12, the result will be an underestimate of the variable locks that can lead to an incomplete search if dual fixings are applied for this variable.

As a special case, the variable locks can be used to detect variables that are not involved in any constraint: that is if, for a variable x_j, $\zeta_j^- = 0$ and $\zeta_j^+ = 0$. If we find such an x_j and the objective functions is monotonically non-decreasing or non-increasing, then Lemma 1.16 can be applied.

Besides detecting isolated variables, the variable locks can also be used to detect isolated constraints: constraints with a variable scope that has no overlap with any other constraint variable scope. Such a constraint defines an independent component and can be solved separately with a specialized algorithm. Such structure appears for several instances of the Miplib 2010 [KAA+11]. For example the instances `bnatt350` contain isolated knapsack constraints which can be solved via dynamic programming.

1.4 Cumulative constraint

This section is dedicated to basics about the global cumulative constraint [AB93]. We first (Section 1.4.1) define this constraint and present in Section 1.4.2 the notation used in this part of the dissertation. In Section 1.4.3 we discuss propagation algorithms for the cumulative constraint. Thereby, we restrict ourselves to those which are realized in the SCIP framework. The discussion for each propagator involves the consistency check and the bound update. The explanations which are needed for an enhanced conflict analysis are discussion in next chapter (Section 2.2.3). Finally, we introduce in Section 1.4.4 an extension of cumulative constraints where jobs are optional.

1.4.1 Definition

In the case of cumulative scheduling, a finite set $\mathcal{J} = \{1, \dots, n\}$ of $n \in \mathbb{N}$ jobs is given which have to be assigned to starting points such that certain conditions are satisfied. These conditions depend on the particular problem. There are usually conditions regarding time which might define a release date, that is the earliest possible start time, and a due date, which is the latest possible completion time. On the other hand, resources are involved which are required by the jobs to be processed. Resources provide a certain capacity. In this work we focus on the following setting: each job $j \in \mathcal{J}$ has a processing time $p_j \in \mathbb{N}$, which is the number of time steps a job runs consecutively after it started. That means, a running job cannot be interrupted (non-preemptive scheduling). Furthermore, each job is equipped with a release date $R_j \in \mathbb{N}$ and a due date $D_j \in \mathbb{N} \cup \{\infty\}$. These two time points define the time window within which a job has to be processed. For job j we call this interval $[R_j, D_j)$ the *feasible time window*. The release date is the earliest possible start point. The due date defines the first time point where the job must not run. Thus, the set of *feasible start points* w.r.t. the release date and due date of job j is given by:

$$T_j = \{t \in \mathbb{N} \mid R_j \leq t \leq D_j - p_j\}.$$

A resource provides a capacity $C \in \mathbb{N}$ which is available during the whole time horizon, i.e., the resource has a renewable capacity. A job has a resource demand $r_j \in \mathbb{N}$ which is consumed in each time step of its processing. Jobs have to be assigned to feasible start points, such that at each point in time the cumulative resource demand is never larger than the capacity C provided by the resource. If $\hat{S}_j \in T_j$ are feasible start times of jobs $j \in \mathcal{J}$, then the following conditions have to hold:

$$\sum_{j \in \mathcal{J}} \mathbb{1}_{[\hat{S}_j, \hat{S}_j + p_j)}(t)\, r_j \leq C \qquad \text{for all } t \in \mathbb{N} \tag{1.2}$$

where the indicator function $\mathbb{1}_M(x)$ evaluates to one if and only if $x \in M$, and zero otherwise. The global cumulative constraint ensures these conditions. Therefore, a cumulative constraint is defined by the start time variables $\boldsymbol{S}$, processing times $\boldsymbol{p}$, resource demands $\boldsymbol{r}$, and its capacity C, i.e.,

$$\texttt{cumulative}(\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}, C).$$

Thereby, the j-element of the vectors $\boldsymbol{S}$, $\boldsymbol{p}$, and $\boldsymbol{r}$, give the required information for job $j \in \mathcal{J}$. The tuple $(\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}, C)$ defines a cumulative constraint uniquely. A cumulative constraint with capacity one is called *disjunctive* constraint [Car82]. In this chapter, however, we are focusing on the more general cumulative constraint with arbitrary capacity.

In this work, we assume that the processing time and resource demand for each job and the resource capacity are fixed values. The only decisions which need to be made are the start times for the jobs.

1.4.2 Notations

Before we discuss several algorithms for the cumulative constraint in more detail let us introduce some notation.

For each job there are some time points of particular interest. These are the earliest start and completion time and the latest start and completion time, respectively. They are defined as follows.

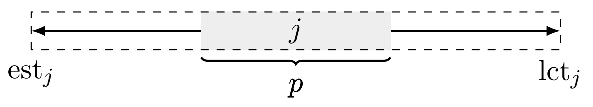
Definition 1.18 (distinctive time points). Given a job j with processing time p_j, release date R_j, and due date D_j, we define the *earliest start time* est_j, the *latest start time* lst_j, the *earliest completion time* ect_j, and the *latest completion time* lct_j as follows:

$$\begin{aligned} \text{est}_j &= R_j & \qquad \text{lst}_j &= D_j - p_j \\ \text{ect}_j &= R_j + p_j & \qquad \text{lct}_j &= D_j. \end{aligned}$$

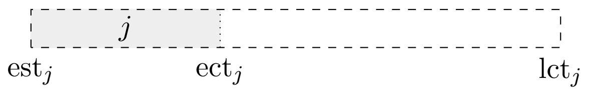
Figure 1.3 shows how jobs are visualized in this dissertation and states the distinctive time points.

The feasible time window $[R_j, D_j) = [\text{est}_j, \text{lct}_j)$[4] of each job j can be used to define the first and latest time points where the resource capacity C can be potentially exceeded. We denote with hmin the minimum time point where the resource capacity could be exceeded and with hmax the minimum time point at which and after which the resource capacity is surely satisfied. Formally:

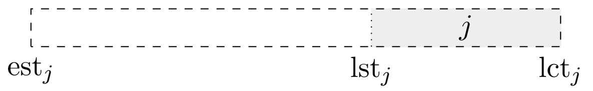
[4]Note that the latest completion time of a job is the first time point where this job definitely does not run.



(a) Visualization of a job j as it is used in this work.



(b) Job j is assigned to its earliest start time (est$_j$) and finishes at its earliest completion time (ect$_j$).



(c) Job j is assigned to its latest start time (lst$_j$) and finishes at its latest completion time (lct$_j$).

Figure 1.3: A job j is visualized with its feasible time window ⬚ given by the earliest start time (est$_j$) and latest completion time (lct$_j$) and with its resource consumption ▨ j defined from the processing time p_j and demand r_j. If the start time is not yet fixed $\longleftrightarrow$ indicates possible placements.

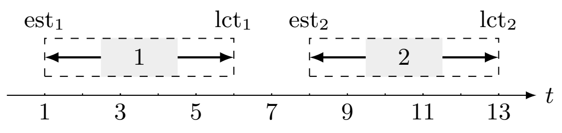
$$\begin{aligned}\text{hmin} &= \inf\{t \in \mathbb{Z} : \sum_{j\in\mathcal{J}} \mathbb{1}_{[\text{est}_j,\text{lct}_j)}(t)\, r_j > C\}\\ \text{hmax} &= \sup\{t \in \mathbb{Z} : \sum_{j\in\mathcal{J}} \mathbb{1}_{[\text{est}_j,\text{lct}_j)}(t-1)\, r_j > C\}.\end{aligned}$$

Note that hmin $\leq$ hmax only holds if hmin and hmax are finite. Example 1.20 shows a setup where this is not the case. Using these two time points the effective time horizon can be defined.

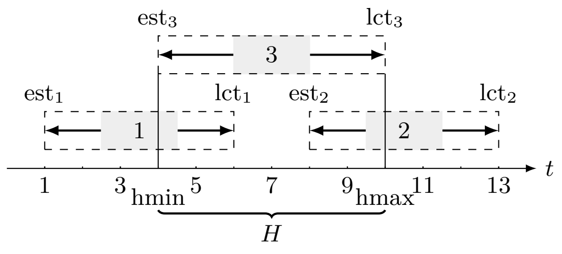
Definition 1.19 (effective horizon). Given a set of jobs $\mathcal{J}$, each with a resource demand $r_j \in \mathbb{N}$ that have to be scheduled on a resource with capacity $C \in \mathbb{N}$. We define the *effective horizon* H as

$$H = \begin{cases} [\text{hmin}, \text{hmax}) & \text{if hmin} < \text{hmax} \\ \varnothing & \text{otherwise.} \end{cases}$$

The effective horizon is a continuous, half-open interval. If the effective horizon is empty, it follows from the definition of hmin and hmax that Condition (1.2) is satisfied for all assignments $\hat{\boldsymbol{S}}$ that respect the release date and due date of each job. Hence, the corresponding resource condition is redundant and the entire cumulative constraint can be removed from the problem instance. On the other hand, if the effective horizon is not empty, it follows that at least one job has an earliest start time which matches hmin and that at least one job has a latest completion time which is equal to hmax. For a disjunctive constraint, we additionally know that hmin defines the first point in time where potentially two jobs are running and hmax the first point in time where this is not the case anymore. The following example illustrates the effective horizon.



(a) In this case hmin $= \infty$, hmax $= -\infty$, and $H = \varnothing$.



(b) Here hmin $= 4$, hmax $= 10$, and $H = [4, 10)$.

Figure 1.4: Illustration of the effective horizon H and the worst case profile for Example 1.20.

Example 1.20. Consider two jobs with unit demand and a resource with unit capacity. The first job has a release date of 1 and a due date of 6. The second job is released at time 8 and has to be completed by time 13. Figure 1.4(a) depicts this situation. In this case hmin $= \infty$ and hmax $= -\infty$ and therefore $H = \varnothing$.

Consider additionally a third job with unit demand. This job has a release date of 4 and a due date of 10. Now the effective horizon is not empty: it is $H = [4, 10)$ (see Figure 1.4(b)).

From the example we can observe that the actual processing time of a job is not relevant for the effective horizon. Only the earliest start times, the latest completion times, the demands, and the available capacity matter. The time points hmin and hmax bound the relevant section of the worst case resource profile [Bec99, BF00b]. The worst case profile returns for each time point the total sum of potentially required capacity. That means, it is assumed that all jobs are processed in every period between their earliest start time and latest completion time (ignoring the actual processing time).

Remark 1.21. The definition of the effective horizon can easily be extended for the case where the available capacity depends on the time point.

The above example also suggests that the corresponding cumulative constraint can be decomposed into two individual cumulative constraints. One handling jobs 1 and 3 and the other jobs 2 and 3. We analyze this observation formally in Section 3.2.

For a cumulative constraint, there are jobs which cannot be processed in parallel, due to their demands. In case of Example 1.20, the available capacity is one which implies that none of the jobs can be processed at the same time. In general, if two jobs have a cumulative demand larger than the given capacity, these jobs have to be arranged sequentially. Such information can be captured in a so-called *non-overlapping* graph. The idea is to capture non-overlapping relation from several cumulative conditions in a single graph. Since the processing times of jobs might differ for different cumulative constraints, the non-overlapping graph depends on the chosen processing times and the analyzed cumulative constraints.

job j	p_j	r_j	R_j	D_j	S_j
1	5	3	1	7	$[1, 2]$
2	4	2	3	10	$[3, 6]$
3	3	2	1	7	$[1, 4]$
4	3	3	1	12	$[1, 9]$
5	3	2	1	7	$[1, 4]$

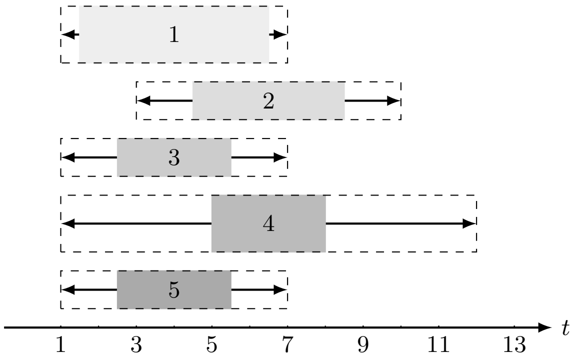


Figure 1.5: The table gives the processing time p_j, the resource demand r_j, the release date R_j, the due date D_j, and the domain of the start time variable S_j for five jobs. The figure visualizes all these information about the five jobs.

Definition 1.22 (non-overlapping graph). Given a set of jobs $\mathcal{J}$, each with a processing time $p_j \in \mathbb{N}$ and a set of cumulative constraints $\mathfrak{C}$, we define the *non-overlapping graph* $G = (\mathcal{J}, E)$ as follows: If and only if there exists a cumulative constraint $\mathcal{C} = (\boldsymbol{S}, \hat{\boldsymbol{p}}, \boldsymbol{r}, C)$ where two jobs i and j are in conflict w.r.t. the capacity C, i.e., $r_i + r_j > C$, $p_i \le \hat{p}_i$, and $p_j \le \hat{p}_j$, then we add an edge between the vertices $i, j \in \mathcal{J}$.

The non-overlapping graph depends on the chosen processing time for each job and can capture information for several cumulative constraints. For the resource-constrained project scheduling problem (see Chapter 4) and the resources allocation and scheduling problem (see Chapter 5) the processing time for a job does not depend on the available resources. Hence, one would choose the unique processing times given for each job to construct a non-overlapping graph. The above definition, however, captures the possibility that the processing time of a job depends on the chosen resource.

Remark 1.23. For a single disjunctive constraint (cumulative constraint with unit capacity) the non-overlapping graph is complete since no pair of jobs can be processed in parallel. Conversely, each clique in a non-overlapping graph implies a disjunctive constraint.

1.4.3 Propagation algorithms

In this section, we present existing propagation algorithms for the cumulative constraint, focusing on those that are available in SCIP. We briefly discuss their individual goals and illustrate the basic idea using an example. Figure 1.5 depicts a set of 5 jobs which we use to visualize the different propagation algorithms. We restrict ourselves to the update of the lower bound of the start time variables. The inference for the upper bound follows by symmetric arguments. In Section 4.3, we present computational results which indicate the impact of the different propagation algorithms in case of resource-constrained project scheduling problems.

In the current version[5] of SCIP we realized three propagation algorithms. These are time-tabling [LL82, KS99], edge-finding [Nui94, Vil09a], and time-tabling edge-finding [Vil11, SFS13]. Besides these propagation algorithms, there exist other propagation algorithms, such as not-first/not-last [Nui94, SW10], extended edge-finding [MH08], and energetic rea-

[5]version 3.0.1.4 or later

soning [BP00, BPN01]. These propagation algorithms are not implemented in SCIP and will not be discussed in this thesis.

Time-tabling

Depending on the tightness of the time window and the processing time of a single job, there might be time points where a job must be executed. These time points define the *core*[6] of a job. If for a job j the latest start time lst_j is smaller than the earliest completion time ect_j, the core γ_j is given by the interval $[\mathrm{lst}_j, \mathrm{ect}_j)$. Otherwise, the core is empty.

The *time-tabling propagator* [LL82, KS99] makes inferences via these core parts of the jobs. Therefore, a *core profile* is defined for any set of jobs $\mathcal{J}$ as follows:

$$\Gamma_{\mathcal{J}} : \mathbb{R} \to \mathbb{N} \text{ with } t \mapsto \sum_{j \in \mathcal{J}} \mathbb{1}_{\gamma_j}(t)\, r_j. \tag{1.3}$$

This function maps each point in time t to the cumulative demand of all cores which need to be processed. Note that this function is a step function. If at any point in time the aggregated demands of the cores exceed the available capacity, an inconsistency is detected.

Lemma 1.24. Given a cumulative constraint $(\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}, C)$. We denote with $\mathcal{J}$ the set of jobs which need to be scheduled. If there exists a time point t with $\Gamma_{\mathcal{J}}(t) > C$, the constraint is not satisfiable.

In case no inconsistency is detected, the core profile can be used to infer bound changes for the start time variables. For a given start time variable S_j the core profile for the remaining jobs $\mathcal{J} \setminus \{j\}$ is scanned for the first potential start time where job j fits w.r.t. the cores of the other jobs.

Lemma 1.25. Given a cumulative constraint $(\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}, C)$. We denote with $\mathcal{J}$ the set of jobs which need to be scheduled. A valid lower bound for the start time variable S_j $(j \in \mathcal{J})$ is:

$$S_j \geq \min\{t \,:\, t \geq \mathrm{est}_j,\ \Gamma_{\mathcal{J}\setminus\{j\}}(\tau) \leq C - r_j \text{ for } \tau = t, \dots, t + p_j - 1\} \tag{1.4}$$

where est_j denotes the earliest start time (lower bound) of the start time variable S_j before the propagation.

The core profile can be constructed in $\mathcal{O}(n \log n)$ where n is the number of jobs. First, all earliest completion and latest start times are collected. After sorting these at most $2n$ time points in non-decreasing order, the profile can be created in linear time. This can be done by iterating over the sorted time points and constructing the step function from the smallest time point to the largest time point. Having this profile, an overload can be detected in linear time by scanning the supporting points of the core profile. This implies that this consistency check can be performed in $\mathcal{O}(n \log n)$. For the update step of the lower bound for a job j, we need to temporarily remove the core of job j from the profile (to retrieve $\Gamma_{\mathcal{J}\setminus\{j\}}$), search for an improved lower bound, and potentially add the core of job j to the profile. Each of these steps requires at most linear time in the number of jobs. Doing this for all jobs, leads to a worst case complexity of $\mathcal{O}(n^2)$ for the bound updates. Recently, a $\mathcal{O}(n \log n)$ version was introduced for the lower bound update for all jobs [OQ13].

[6] also called compulsory part

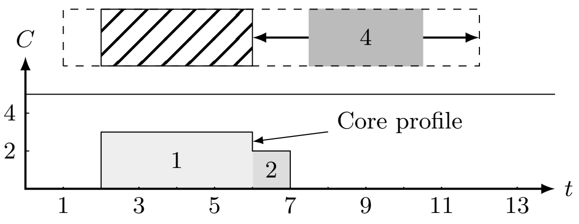


Figure 1.6: Visualization of the core profile induced by the jobs stated in Figure 1.5. The pattern ▨ indicates the time interval where job 4 cannot be scheduled due to required demands of the cores.

We refer to [LL82, KS99, BPN01, OQ13] for a more formal description and analysis of this propagator. The following examples illustrate the basic idea.

Example 1.26. Consider the jobs which are shown in Figure 1.5. The core profile is depicted in Figure 1.6. Note that jobs 3, 4, and 5 have empty cores. This profile gives the load w.r.t. the time which needs to be processed for sure. Assume a resource capacity of 5 as shown in Figure 1.6. This core profile does not imply an inconsistency since the peak is 3. However, this profile can be used to infer a larger lower bound for job 4. This job has a demand of 3 and so it cannot be processed in the interval $[2, 6)$ since the core profile (of the remaining jobs) proves that at any point in time within this interval less than 3 units of the resource capacity are available. Since job 4 has a processing time of 3 and a release date of 1, it cannot be scheduled before the conflicting interval $[2, 6)$. Hence, the release date of job 4 can be updated to 6 which is also given by Formula (1.4). Figure 1.6 visualizes the lower bound update step. After improving the lower bound of the start time variable belonging to job 4, this job still has an empty core. For the remaining jobs a bound update cannot be deduced from the core profile since at any point in time at least 2 resource units are available.

A version of the time-tabling propagation algorithm posts bound changes incrementally by changing a bound by at most the processing time of the corresponding job at a time. This does not give any disadvantages w.r.t. time-tabling propagation. For this purpose, the update step (1.4) of Lemma 1.25 is replaced by:

$$S_j \geq \min\{t\,:\,\mathrm{est}_j \leq t \leq \mathrm{ect}_j,\ \Gamma_{\mathcal{J}\setminus\{j\}}(\tau) \leq C - r_j \text{ for } \tau = t, \ldots, t + p_j - 1, \}. \qquad (1.5)$$

This condition bounds the lower bound update from above with the earliest completion time ect_j.

Example 1.27 (Continuing Example 1.26). In Example 1.26 we illustrated the time-tabling propagator and showed that for the start time variable S_4 a lower bound update from 1 to 6 can be inferred. This increase of the lower bound is larger than the processing time of job 4 (which is 3). If the incremental update would be applied, the lower bound is increased in two steps. These are $[\![S_4 \geq 4]\!]$ followed by $[\![S_4 \geq 6]\!]$.

Edge-finding

In contrast to the time-tabling propagator the *edge-finding* [Nui94, Vil09a] algorithm does not use any direct placement arguments. It reasons about the required energies of the individual jobs. The energy of a job is the product of the processing time and the resource demand. That is $p_j \cdot r_j$ for job j. For a non-empty time interval $[a, b)$ the energy of all jobs which have to be processed in that interval are aggregated. If the resulting total energy is larger than the available energy, an inconsistency is discovered. Otherwise, bound changes can be inferred. Thereby, this propagator focuses only on jobs which have to be completely processed in that particular time interval, i.e., the earliest start times have to be larger or equal to a and the latest completion times smaller or equal to b. We define $E_{\mathcal{J}}(a, b)$ to be the energy of all jobs $\mathcal{J}$ which have to be processed completely in the time interval $[a, b)$. That is:

$$E_{\mathcal{J}}(a,b) = \sum_{j \in \mathcal{J}} e_j(a,b) \quad \text{with} \quad e_j(a,b) = \begin{cases} p_j \cdot r_j & \text{if } [\text{est}_j, \text{lct}_j) \subseteq [a,b) \\ 0 & \text{otherwise.} \end{cases} \tag{1.6}$$

Having this notation, the inconsistency check and the lower bound update step can be formalized as follows.

Lemma 1.28. Given a cumulative constraint $(\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}, C)$, an interval $[a, b)$ with $a < b$, and the set of jobs $\mathcal{J}$ which need to be scheduled. The constraint is not satisfiable if $E_{\mathcal{J}}(a, b) > (b - a) \cdot C$.

If no inconsistency is inferred, the energy within this interval can be used to retrieve bound updates for jobs which do not contribute to the aggregated energy, i.e., jobs $j \in \mathcal{J}$ with $e_j(a, b) = 0$.

Lemma 1.29. Given a cumulative constraint $(\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}, C)$ and an interval $[a, b)$ with $a < b$ and $E_{\mathcal{J}}(a, b) \leq (b-a) \cdot C$. Let us denote with $\mathcal{J}$ the set of jobs which need to be scheduled. If for a job j, it holds that $e_j(a, b) = 0$ and

$$E_{\mathcal{J}}(a,b) + r_j \cdot (\min\{b, \text{ect}_j\} - \max\{a, \text{est}_j\}) > (b-a) \cdot C, \tag{1.7}$$

the lower bound of the start time variable S_j can be bounded by:

$$S_j \geq b - \left\lfloor \frac{1}{r_j} \left((b-a) \cdot C - E_{\mathcal{J}}(a,b) \right) \right\rfloor. \tag{1.8}$$

If the cumulative constraint is not inconsistent w.r.t. the time interval $[a, b)$, Condition (1.7) implies that job j overlaps with the interval $[a, b)$ if it is scheduled at its earliest start time. This holds since $E_{\mathcal{J}}(a, b) \leq C \cdot (b-a)$ implies $r_j \cdot (\min\{b, \text{ect}_j\} - \max\{a, \text{est}_j\}) > 0$. Otherwise, Condition (1.7) is not satisfied. Furthermore, the earliest start time is not feasible due to a overload within the time interval $[a, b)$. Hence, part of the job needs to be processed after the interval $[a, b)$ which is enforced by Inequality (1.8). Due to a lack of energy for job j within the inspected interval, it follows that the maximum potential overlap of job j is smaller after the propagation took place. That is:

$$\begin{aligned} \left\lfloor \frac{1}{r_j} \left((b-a) \cdot C - E_{\mathcal{J}}(a,b) \right) \right\rfloor &\leq \frac{1}{r_j} \left((b-a) \cdot C - E_{\mathcal{J}}(a,b) \right) \\ &\overset{(1.7)}{<} \min\{b, \text{ect}_j\} - \max\{a, \text{est}_j\}. \end{aligned} \tag{1.9}$$

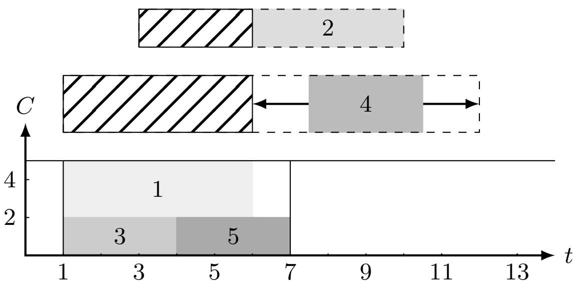


Figure 1.7: This figure shows the lower bound update due to the edge-finding propagator for jobs 2 and 4 w.r.t. the time interval $[1, 7)$. The pattern ▨ indicates the time interval where job 2 and 4 cannot be scheduled.

Currently, the best known worst case complexity for this propagator is $\mathcal{O}(kn \log n)$ where n is the number of jobs and k the capacity of the cumulative constraint. For a more detailed description and analysis of this propagator, we refer to [Vil09b, Sco10]. In SCIP we implemented a version which has a worst case complexity of $\mathcal{O}(n^2 \log n)$ (see [Vil09a]). The following example illustrates the bound update step of the edge-finding propagator.

Example 1.30. Consider the jobs of our running example (Figure 1.5) and the time interval $[1, 7)$. Within this time interval jobs 1, 3, and 5 have to be processed since their release and due dates are 1 and 7, respectively. In total these three jobs consume 27 units of the available energy which is 30. This means that only 3 units are available for the remaining jobs within the interval $[1, 7)$. Scheduling either job 2 or 4 at its earliest start time would lead to an overload within the time interval $[1, 7)$. Condition (1.7) of Lemma 1.29 is satisfied for both jobs. Hence, the lower bounds for job 2 and 4 can be propagated. Following Formula (1.8), the start time variables for these two jobs can be bounded from below:

$$S_2 \geq 7 - \left\lfloor \frac{1}{2}\left((7-1)\cdot 5 - 27\right) \right\rfloor = 6 \qquad S_4 \geq 7 - \left\lfloor \frac{1}{3}\left((7-1)\cdot 5 - 27\right) \right\rfloor = 6.$$

Figure 1.7 depicts the lower bound update for jobs 2 and 4.

Time-tabling edge-finding

The last propagation algorithm which we discuss is called *time-tabling edge-finding* [Vil11, SFS13]. As the name suggests, it is a combination of the two previously introduced propagation algorithms: time-tabling and edge-finding. The basic idea is to add some placement arguments to the classical edge-finding algorithm. The edge-finding algorithm only considers jobs which have to be processed completely within the interval under investigation. The time-tabling edge-finding propagator attempts to use the information of the core profile to add additional fixed energy consumptions. Thereby, the worst case complexity should not be increased compared to the edge-finding propagation algorithm.

To be able to take advantages of the time-tabling and edge-finding idea, the processing time of each job is split into a fixed and free part. The fixed part contains the portion of the job which is covered by the core of a job (this can be zero). The remaining processing time belongs to the free part, meaning it is not clear where it will be placed exactly. More formally, for a job j we denote with p_j^{TT} and p_j^{EF} the fixed and free part of job j, respectively. That is:

$$p_j^{TT} = \max\{0, \mathrm{ect}_j - \mathrm{lst}_j\} \qquad p_j^{EF} = p_j - p_j^{TT}.$$

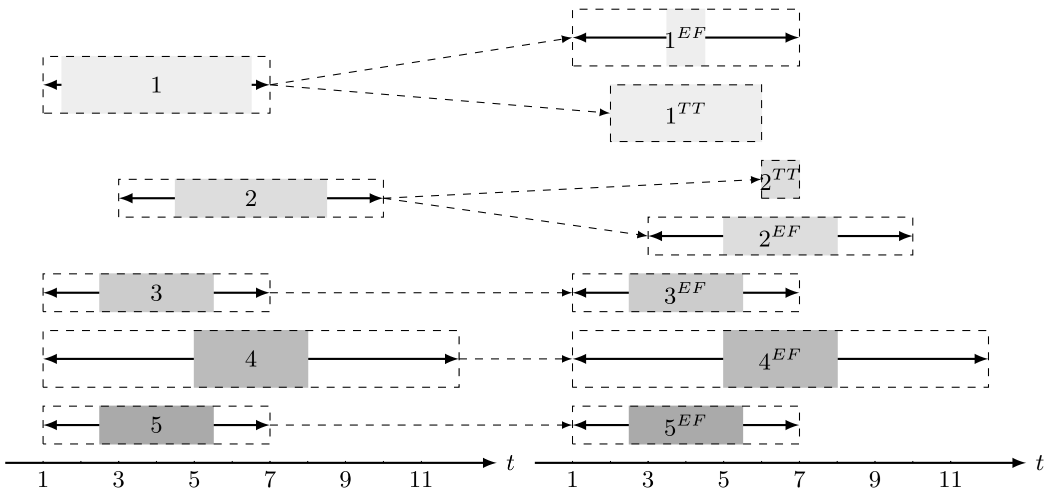


Figure 1.8: Visualization of the fixed and free part for the jobs belonging to the running example (see Figure 1.5). Each job j is divided into two jobs. One representing the fixed part j^{TT} and the other representing the free part j^{EF}. In case of jobs 3, 4, and 5 the fixed part is empty, therefore, these parts are not printed.

Figure 1.8 depicts the fixed and free part of the processing time for the jobs of our running example.

For a given time interval, the time-tabling edge-finding propagation algorithm collects the energy which is given by the free parts of the jobs and adds the contribution of the cores which are in the interval under investigation. For the latter, the core profile is used to compute for each point in time t the total energy of all cores at time point t and later. That is for a given set of jobs $\mathcal{J}$:

$$Vol_{\mathcal{J}} : \mathbb{R} \to \mathbb{R} \text{ with } t \mapsto \int_t^{\infty} \Gamma_{\mathcal{J}}(\tau)\, \mathrm{d}\tau.$$

Having the core profile $\Gamma_{\mathcal{J}}$, which is a step function, $Vol_{\mathcal{J}}$ can be computed for the supporting point of the core profile in linear time w.r.t. the number of jobs. We define the core energy within a non-empty interval $[a, b)$ with $E^{TT}_{\mathcal{J}}(a, b)$. It is given by:

$$E^{TT}_{\mathcal{J}}(a, b) = Vol_{\mathcal{J}}(a) - Vol_{\mathcal{J}}(b).$$

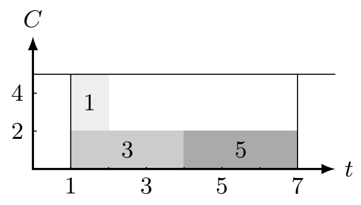
The core energy is calculated in constant time if $Vol_{\mathcal{J}}$ is known for the supporting points of the core profile.

For a set of jobs $\mathcal{J}$ and an interval $[a, b)$ the contribution of the fixed parts of the jobs is covered by $E^{TT}_{\mathcal{J}}(a, b)$. For the free part we define:

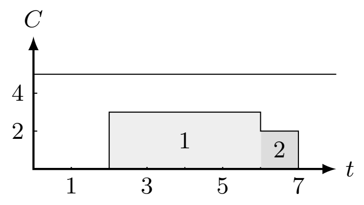
$$E^{EF}_{\mathcal{J}}(a, b) = \sum_{j \in \mathcal{J}} e^{EF}_j(a, b) \quad \text{with} \quad e^{EF}_j(a, b) = \begin{cases} p^{EF}_j \cdot r_j & \text{if } [\mathrm{est}_j, \mathrm{lct}_j) \subseteq [a, b) \\ 0 & \text{otherwise.} \end{cases}$$

Note that this definition is similar to the one for $E_{\mathcal{J}}(a, b)$ (see Equation (1.6)) which is needed for the edge-finding propagator. It only differs in using the processing time of the free parts of the jobs to compute the energy contribution of a job, whereas for $E_{\mathcal{J}}(a, b)$ the complete processing time is considered.

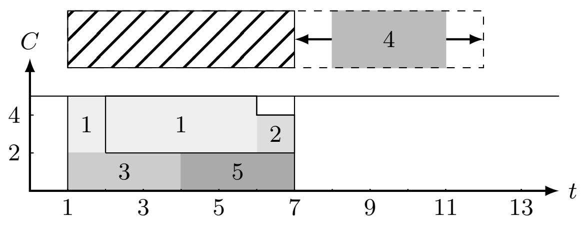
After separating the contribution of a job into a free and fixed part and defining the corresponding energies for a certain time interval, we can formalize the time-tabling edge-finding consistency check and lower bound update.



(a) Job 1, 3, and 5 add energy with their free part within the time window $[1, 7)$.



(b) Job 1 and 2 contribute to the core profile.



(c) Combining core profile and energy consumption of the free parts lead to lower bound update for job 4.

Figure 1.9: This figure shows the lower bound update due to the time-tabling edge-finding propagation algorithm for job 4 w.r.t. the time interval $[1, 7)$ (Example 1.33). Jobs 1, 3, and 5 contribute via the classical edge-finding argument with their free part (Figure (a)). Additionally, job 1 and job 2 contribute with their cores (Figure (b)). These contributions imply a lower bound of 7 for the start time variable belonging to job 4. The pattern ▨ indicates the time interval where job 4 cannot be scheduled (Figure (c)).

Lemma 1.31. Given a cumulative constraint $(\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}, C)$ and an interval $[a, b)$ with $a < b$. Let us denote with $\mathcal{J}$ the set of jobs which need to be scheduled. The constraint is not satisfiable if $E^{TT}_{\mathcal{J}}(a, b) + E^{EF}_{\mathcal{J}}(a, b) > (b - a) \cdot C$.

Lemma 1.32. Given a cumulative constraint $(\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}, C)$ and an interval $[a, b)$ with $a < b$. Let us denote with $\mathcal{J}$ the set of jobs which need to be scheduled. If for a job j it holds that $e^{EF}_j(a, b) = 0$ and

$$E^{EF}_{\mathcal{J}}(a, b) + E^{TT}_{\mathcal{J}\setminus\{j\}}(a, b) + r_j \cdot (\min\{b, \mathrm{ect}_j\} - \max\{a, \mathrm{est}_j\}) > (b - a) \cdot C,$$

the lower bound of the start time variable S_j can be bounded by

$$S_j \geq b - \left\lfloor \frac{1}{r_j} \left((b - a) \cdot C - E^{EF}_{\mathcal{J}}(a, b) - E^{TT}_{\mathcal{J}\setminus\{j\}}(a, b) \right) \right\rfloor .$$

The requirement $e^{EF}_j(a, b) = 0$ ensures that job j has no contribution to $E^{EF}_{\mathcal{J}}(a, b)$. In case of the core energy, we need to make sure that the core of job j is removed. This is captured by $E^{TT}_{\mathcal{J}\setminus\{j\}}(a, b)$. The following example visualizes this lower bound update step.

Example 1.33. Recalling our running example (see Figure 1.5) and focusing on the time window $[1, 7)$, Figure 1.9 illustrates the lower bound update for job 4 using the time-tabling edge-finding propagation algorithm. Since job 3 and 5 have an empty core, their energy is completely captured by the energy consumption of the free parts (see Figure 1.9(a)). On the other hand, job 2 only contributes to its core. Job 1, however, has a core and lies complete within the time interval $[1, 7)$. Hence, its energy is split. The energy induced by the core is part of the core profile. The remaining energy is added to the energy of the free parts. Job 1 contributes its complete energy (see Figure 1.9(c)). The total energy measured for the time window $[1, 7)$ is 29 units:

$$E^{EF}_{\{1,2,3,4,5\}}(1, 7) = 15 \qquad E^{TT}_{\{1,2,3,5\}}(1, 7) = 14.$$

Having this, it follows that job 4 cannot start before time point 7. Note that this is a larger lower bound than infered by the time-tabling and the edge-finding propagation algorithm for this variable (see Examples 1.26 and 1.30).

The basic time-tabling edge-finding algorithm can be extended to incorporate even more energy consumptions. Jobs which do not participate in $E^{EF}_{\mathcal{J}}(a, b)$ since they do not lie completely within the time interval $[a, b)$ but overlap with the interval on the boundary could potentially contribute with their free part. This contribution can be easily included in the algorithm without increasing its worst case complexity of $\mathcal{O}(n^2)$ where n denotes the number of jobs. For more details about this extension and a complete analysis of the time-tabling edge-finding propagation algorithm we refer to [Vil11, SFS13].

1.4.4 Optional jobs

One possible extension of the cumulative constraint allows for jobs to be optional. Jobs can be processed by different resources and depending on the resources a job might have different demands and processing times. If the decision is made that a job needs to be processed on a certain resource, the assigned job needs be considered and it needs to be ensured that the job is processed such the capacity of the resource is respected. The resource allocation and scheduling problem introduced in Chapter 5 contains such characteristics.

An *optional cumulative constraint* which captures this extension can be defined formally as follows. Given a finite set $\mathcal{J} = \{1, \dots, n\}$ of $n \in \mathbb{N}$ jobs. As in the cumulative constraint case, each job $j \in \mathcal{J}$ has a release date $R_j \in \mathbb{N}$ and a due date $D_j \in \mathbb{N} \cup \{\infty\}$. Furthermore, each job has a resource demand $r_j \in \mathbb{N}$ and a processing time $p_j \in \mathbb{N}$. The processing time defines for how many consecutive time points for which a job is processed after it has started. During the processing of a job it consumes at each time point the resource demand r_j if it is assigned to the resource. To capture the aspect of optional jobs, a binary variable x_j for each job is given. If x_j is 1 the job is assigned to the resource and needs to be considered. In case of x_j equals 0 job j can be ignored. A resource provides a capacity $C \in \mathbb{N}$ which is available during the whole time horizon. Jobs which are assigned to a resource need to be scheduled to a feasible start point, such that for all points in time, the cumulative resource demand is never larger than the capacity C provided by the resource. If $\hat{S}_j \in T_j$ is a feasible start time of job j and $\hat{x}_j$ chosen for job j, then the following conditions have to hold:

$$\sum_{j \in \mathcal{J}} \mathbb{1}_{t \in [\hat{S}_j, \hat{S}_j + p_j)}(t)\, r_j \cdot \hat{x}_j \leq C \qquad \text{for all } t \in \mathbb{N} \tag{1.10}$$

where the indicator function is defined as $\mathbb{1}_M(x) = 1$ if $x \in M$, and zero otherwise. The tuple $(\boldsymbol{x}, \boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}, C)$ defines an optional cumulative constraint uniquely. The j-element of the vectors $\boldsymbol{x}$, $\boldsymbol{S}$, $\boldsymbol{p}$, and $\boldsymbol{r}$ represent the information for job $j \in \mathcal{J}$. In models we use the notation

$$\texttt{optcumulative}(\boldsymbol{x}, \boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}, C).$$

An optional cumulative constraint where all choice variables are fixed to one or zero can be transformed into a cumulative constraint.

Known propagation algorithms for the cumulative constraint without optional jobs can be easily adapted to the case where optional jobs are present [BF00a]. The basic idea is as follows. We propagate all jobs that are known to execute on the resource with standard cumulative propagation (e.g., the propagation algorithms introduced in the previous sections). This has the advantage that if the propagation algorithms for the cumulative constraints are improved, the optional cumulative constraint directly benefits from this improvement. In addition, for each job j that is still optional, we perform singleton arc-consistency [DB97]. We assume that the job will execute on the resource and trigger propagation.[7] If the propagation reaches a dead-end, we can soundly conclude that the job cannot execute on the resource and appropriately set the x_j variable to zero. Otherwise, we retain the pruned domain for the start time variable S_j variable if the optional cumulative constraint is the only one locking the start time variable. In any case, the domains of all other variables are restored. Note that, this propagation is stronger, but more costly, than the propagation of cumulative constraints with optional jobs discussed in [Vil05].

[7]Singleton arc-consistency is similar but more general than the shaving technique in the scheduling literature [MS96].

2 Conflict relaxations and linear relaxations for cumulative constraints

The linear relaxation of a MIP, see Section 1.2.2, is one of the most important features in computational mixed-integer programming. Any feasible solution for this relaxation satisfies all linear constraints but might violate some of the integrality conditions. This relaxation often provides a good lower bound on the optimal objective value of the corresponding MIP. In addition, it is typically used to make branching decisions. In this chapter we define the *conflict relaxation*. This relaxation plays an important role for conflict-driven SAT solvers as it is dynamically tightened during the search due to the analysis of infeasible sub-problems. It provides additional inferences to help pruning other sub-problems. SAT solvers that exploit the conflict relaxation (i.e., conflict-directed clause learning (CDCL) solvers [BHvMW09]) also use statistics based on this relaxation to make branching decisions.

In this chapter, we focus on conflict relaxations and linear relaxations in the context of cumulative constraints. In particular we analyze a special class of propagation algorithms which we call *energy-based propagation algorithms*. The propagation algorithms introduced in Section 1.4.3 belong to this class. We discuss how algorithms belonging to this class contribute to the analysis of infeasible sub-problems and can be used to construct linear relaxations for the cumulative constraint with optional jobs. Both contributions are based on the same idea.

Contribution. In Section 2.1 we formally define *energy-based propagation algorithms*, a class of propagation algorithms for cumulative constraints. These algorithms use specific energy-based arguments to infer bound changes or to detect infeasibilities. For this class of propagation algorithms we develop explanations (see Definition 2.6) for their inferences (see Section 2.2.3), generalizing previously published explanations. These explanations can be then used to generate conflict constraints during analysis of infeasible sub-problems. The propagation algorithms introduced in Section 1.4.3 are an example of energy-based propagation algorithms. For these algorithms we apply the developed concept for energy-based propagation algorithms and present the implied explanations which are a superset of previously published explanations.

For the cumulative constraint with optional jobs, we develop new linear relaxations in Section 2.3.2. We prove that each propagation algorithm belonging to the class of energy-based propagation algorithm implies a linear relaxation. This general concept comprises the known linear relaxations for the cumulative constraint with optional jobs. In addition we apply this method and present a new linear relaxation implied by the energetic reasoning propagation algorithm.

Outline. This chapter is divided into four sections. In Section 2.1 we define the class of energy-based propagation algorithms for cumulative constraints and show that the propagation algorithms introduced in Section 1.4.3 belong to this class of propagation algorithms.

Section 2.2 focuses on the conflict relaxation. We first recall, in Section 2.2.1, the general concept of conflict analysis which is used to analyze infeasible sub-problems and return conflict constraints that form a conflict relaxation. In addition we describe conflict-driven search for binary variables. In Section 2.2.2 we adapt the conflict-driven search idea to variables with integer domains. In Section 2.2.3 we develop explanations for energy-based propagation algorithms. Explanations are needed during the conflict analyze to retrieve conflict constraints. We apply the concept of explanations for energy-based propagation algorithms to the propagation algorithms introduced in the previous chapter. We demonstrate that the introduced explanations generalize known explanations for the considered algorithms. Section 2.3 is dedicated to linear relaxation for a cumulative constraint with optional jobs. We first recall, in Section 2.3.1, known linear relaxations for this structure. In Section 2.3.2 we develop a method which constructs a linear relaxation for each propagation algorithm belonging to the class of energy-based propagation algorithms. We show that two previously published linear relaxations fit into this concept and apply this method to develop a new linear relaxation based on the energetic reasoning propagation algorithm. We close this chapter with Section 2.4 by summarizing the two concepts developed for the conflict and the linear relaxation for energy-based propagation algorithms.

2.1 Energy-based propagation algorithms

In this section we define a class of propagation algorithms for the cumulative constraint which we call *energy-based propagation algorithms*. In the following sections we develop general methods w.r.t. the conflict relaxation and linear relaxation, respectively, for this class of propagation algorithms.

Most propagation algorithms for the cumulative constraint are based on a volume argument to detect inconsistency or to infer domain changes. For a non-empty interval $[a, b)$ it is known how much energy is available. That is $(b - a) \cdot C$ where C denotes the finite capacity of the cumulative constraint at each time point. For each job j a lower bound $\underline{e}_j(a, b)$ on the energy consumption within this interval is constructed. That means in any feasible solution (that satisfies the variable domains which serve as input for the propagation algorithm), a job consumes at least this amount of energy in the interval $[a, b)$. Using purely the boundary of a job the lower bound $\underline{e}_j(a, b)$ for a fixed interval $[a, b)$ can be bounded from above:

$$\underline{e}_j(a, b) \leq e'_j(a, b) = \max\{0, \min\{b - a, p_j, \mathrm{ect}_j - a, b - \mathrm{lst}_j\}\} \cdot r_j \tag{2.1}$$

where ect_j and lst_j denote the earliest completion time and the latest start time of job j, respectively, and p_j and r_j the processing time and resource demand. Note that $e'_j(a, b)$ is the lower bound for the energy consumption used by the energetic reasoning propagation algorithm [BPN01].

Having for each job a lower bound for the energy consumption, a consistency check and variable bound improvements can be performed by volume formulas.

Inconsistency check. An inconsistency in interval $[a, b)$ is detected if the sum of the lower bounds on the energy consumption for this interval $\underline{e}_j(a, b)$ is larger than the available energy:

$$\sum_{j \in \mathcal{J}} \underline{e}_j(a, b) > (b - a) \cdot C. \tag{2.2}$$

Domain change. An improved earliest start time due to interval $[a,b)$ for a job j with resource demand r_j can be inferred if it is infeasible to schedule this job at its current earliest start time. This means that the energy consumption in the interval $[a,b)$ would increase above the available energy. This can be formalized by the following two conditions. First, there must not be an overload within the interval $[a,b)$ if job j is ignored:

$$\sum_{i\in\mathcal{J}\setminus\{j\}} \underline{e}_i(a,b) \le (b-a)\cdot C. \tag{2.3}$$

If this is not the case there is an inconsistency. Second, scheduling job j at its earliest start time est_j leads to an overload in the corresponding interval:

$$\sum_{i\in\mathcal{J}\setminus\{j\}} \underline{e}_i(a,b) + r_j\cdot(\min\{b,\mathrm{ect}_j\} - \max\{a,\mathrm{est}_j\}) > (b-a)\cdot C. \tag{2.4}$$

These two conditions imply $\min\{b,\mathrm{ect}_j\} - \max\{a,\mathrm{est}_j\} > 0$. When these conditions hold, the earliest start time for job j can be bounded from below by:

$$b - \left\lfloor \frac{1}{r_j}\left((b-a)\cdot C - \sum_{i\in\mathcal{J}\setminus\{j\}} \underline{e}_i(a,b)\right)\right\rfloor \le S_j. \tag{2.5}$$

Symmetrical arguments can be used to bound the latest completion time of a job from above.

The algorithms which follow this scheme differ in the computation of the lower bound for the energy consumption of a fixed interval and the type of intervals which are considered. The following definition states properties which subsume the energy-based propagation algorithms.

Definition 2.1 (energy-based propagation algorithms). A propagation algorithm for the cumulative constraint (as defined in Section 1.4.1) belongs to the class of energy-based propagation algorithms if it uses for each job j a lower bound $\underline{e}_j(a,b)$ on the energy consumption for a fixed non-empty interval $[a,b)$ which is not larger than

$$e'_j(a,b) = \max\{0, \min\{b-a, p_j, \mathrm{ect}_j - a, b - \mathrm{lst}_j\}\}\cdot r_j \tag{2.6}$$

and uses Inequality (2.2) and Inequality (2.5) (with Conditions (2.3) and (2.4)) to detect an inconsistency or an lower bound on the earliest start time, respectively.

The propagation algorithms introduced in Section 1.4.3 belong to this class. In the following we discuss how these algorithms fit into this class.

Time-tabling. The time-tabling propagation algorithm considers intervals of length one. For each job j the core γ_j (see Section 1.4.3) is used to determine a lower bound for the energy consumption within the considered unit interval. This lower bound is r_j if the core overlaps with the interval or zero, otherwise. The core profile $\Gamma_{\mathcal{J}}(t)$ (see Equation (1.3)) is used to capture a lower bound for the energy consumption for each point in time t:

$$\Gamma_{\mathcal{J}}(t) \overset{\overset{(1.3)}{\downarrow}}{=} \sum_{j\in\mathcal{J}} \mathbb{1}_{\gamma_j}(t)\, r_j = \sum_{j\in\mathcal{J}} \underline{e}_j(t,t+1) \quad \text{with} \quad \underline{e}_j(t,t+1) = \begin{cases} r_j & \text{if } \mathrm{lst}_j \le t < \mathrm{ect}_j \\ 0 & \text{otherwise.}\end{cases}$$

Clearly, $\underline{e}_j(t,t+1) = e'_j(t,t+1)$. Hence, this propagation algorithm belongs to the class of energy-based propagation algorithms if the bound update is done w.r.t. Inequality (1.5).

Edge-finding. For a fixed interval $[a,b)$ the edge-finding propagation algorithm uses $e_j(a,b)$ to compute a lower bound on the contribution of a job j for this interval. This energy lower bound $e_j(a,b)$ is the overall energy requirement of a job j (demand multiplied by the processing time) if the job needs to be executed entirely in the interval $[a,b)$ in any case, otherwise, it is zero:

$$\underline{e}_j(a,b) = e_j(a,b) \overset{(1.6)}{\overset{\downarrow}{=}} \begin{cases} p_j \cdot r_j & \text{if } [\text{est}_j, \text{lct}_j) \subseteq [a,b) \\ 0 & \text{otherwise.} \end{cases}$$

The intervals of interest are defined by the earliest start time and latest completion time of each job. Note that $e_j(a,b) \leq e'_j(a,b)$.

Time-tabling edge-finding. The time-tabling edge-finding propagation algorithm splits each job into a fixed and a free part. For the fixed parts the cores of the jobs are used to construct a lower bound on the energy consumption within a fixed interval $[a,b)$:

$$e_j^{TT}(a,b) = \int_a^b \mathbb{1}_{\gamma_j \cap [a,b)}(t)\,\mathrm{d}t \cdot r_j$$

where γ_j defines the core of job j. The lower bound used for the edge-finding propagation algorithm is taken to bound the energy consumption for the free part:

$$e_j^{EF}(a,b) = \begin{cases} p_j^{EF} \cdot r_j & \text{if } [\text{est}_j, \text{lct}_j) \subseteq [a,b) \\ 0 & \text{otherwise} \end{cases}$$

with $p_j^{EF} = p_j - \max\{0, \text{ect}_j - \text{lst}_j\}$. See Section 1.4.3 for more details. For each job j, the sum $e_j^{TT}(a,b) + e_j^{EF}(a,b)$ is a lower bound on the whole energy consumption for this job within interval $[a,b)$. As for the edge-finding propagation algorithm, the intervals of interest are defined by the earliest start time and by the latest completion time of each job.

The total energy contribution considered by the time-tabling edge-finding propagation algorithm for any job j is not larger than $e'_j(a,b)$ (see Equation (2.1)).

Lemma 2.2. Given an interval $[a,b)$ with $a < b$ and a job j with processing time $p_j > 0$ and resource demand $r_j > 0$, it holds that:

$$e_j^{TT}(a,b) + e_j^{EF}(a,b) \leq e'_j(a,b).$$

Proof. Given an interval $[a,b)$ with $a < b$ and a job j with processing time p_j and resource demand $r_j > 0$ if $e'_j(a,b) = 0$ it can be easily shown that $e_j^{EF}(a,b) = e_j^{TT}(a,b) = 0$. Hence, it remains to be shown that the claim holds for $e'_j(a,b) > 0$. To prove this we perform a case distinction over the elements of the minimum in Equation (2.6) defining $e'_j(a,b)$.

Case $e'_j(a,b) = (b-a) \cdot r_j$: Hence, job j will be processed during the whole time interval $[a,b)$ in any case. It follows, $\text{lst}_j \leq a$, $\text{ect}_j \geq b$, and $p_j \geq b - a$. This implies:

$$e_j^{TT}(a,b) = (b-a) \cdot r_j$$

In addition we know that

$$[a,b) \subseteq [\text{lst}_j, \text{ect}_j) \subseteq [\text{est}_j, \text{lct}_j)$$

If $[\text{lst}_j, \text{ect}_j) = [\text{est}_j, \text{lct}_j)$ it follows that $p_j^{EF} = 0$. Hence, $e_j^{EF}(a,b) = 0$. If $[\text{lst}_j, \text{ect}_j) \subset [\text{est}_j, \text{lct}_j)$ it follows that $[a,b) \subset [\text{est}_j, \text{lct}_j)$. Hence, $e_j^{EF}(a,b) = 0$ by definition which proves the claim for this case.

Case $e'_j(a,b) = p_j \cdot r_j$: Since $p_j \cdot r_j$ bounds $e_j^{TT}(a,b) + e_j^{EF}(a,b)$ from above, the claim holds.

Case $e'_j(a,b) = (b - \text{lst}_j) \cdot r_j$: We assume that $(b - \text{lst}_j) < p_j$. Otherwise, $(b - \text{lst}_j) = p_j$ and Case 2 proves the claim. This assumption implies that $b < \text{lct}_j$ which gives $[\text{est}_j, \text{lct}_j) \nsubseteq [a,b)$. It follows by definition $e_j^{EF}(a,b) = 0$.

Further, we assume $\text{lst}_j > a$. If $\text{lst}_j = a$, then Case $e'_j(a,b) = (b-a) \cdot r_j$ proves the claim. From which follows that,

$$\gamma_j \cap [a,b) \subseteq [\text{lct}_j, b).$$

This bounds $e_j^{TT}(a,b)$ from above with $(b - \text{lst}_j) \cdot r_j$ which proves the claim.

Case $e'_j(a,b) = (\text{ect}_j - a) \cdot r_j$: Arguments symmetric to the previous case give the proof this case. $\square$

Hence, this propagation algorithm is an energy-based propagation algorithm.

Energetic reasoning. The energetic reasoning propagation algorithm [BPN01] uses the same intervals as the edge-finding propagation algorithm and the time-tabling edge-finding propagation algorithm. For a fixed interval $[a,b)$ it bounds the energy consumption of each job by:

$$\underline{e}_j(a,b) = e'_j(a,b) = \max\{0, \min\{b-a, p_j, \text{ect}_j - a, b - \text{lst}_j\}\} \cdot r_j$$

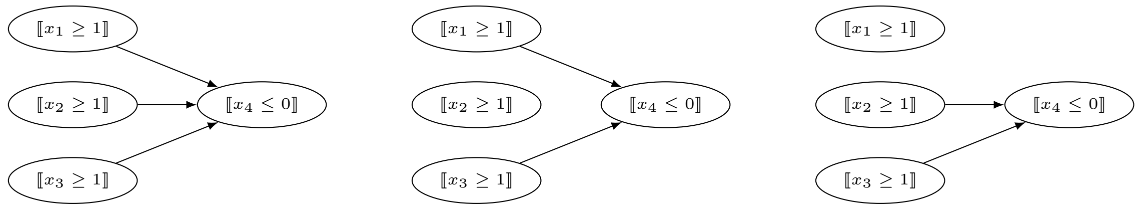
which equals Equality (2.6). Hence, the energetic reasoning propagation algorithm is an energy-based propagation algorithm.

In the remainder of this chapter we develop general methods w.r.t. the conflict relaxation and linear relaxation for the class of energy-based propagation algorithms.

2.2 Conflict relaxation

During a branch-and-bound tree search, infeasible sub-problems can occur. That means the problem at a search node does not admit any feasible solutions. Infeasible sub-problems can be analyzed to learn constraints which might help to subsequently prune the search tree. These constraints need to be fulfilled by any feasible solution and therefore form a relaxation of the original problem which we call the *conflict relaxation*. The technique of analyzing infeasible sub-problems, called *conflict analysis*. It has been considered for COP [SS77], SAT [MSS99], and MIP [Ach07a, SS06] and is synonymously known as *nogood learning*. In addition, statistics about the distribution of variables in conflict constraints can be used to drive the search. In the SAT community this is known as conflict-driven search [MMZ$^+$01]. For solving mixed-integer programs such statistics have been used in combination with other statistics to decide how to branch [AB09].

For cumulative scheduling problems such as the resource-constrained project scheduling problem (see Chapter 4) it has been shown by empirical studies [SFSW11, HS11, Sch12, SFS13, SFSW13] that a conflict relaxation can be crucial to solve these problems efficiently. A state-of-the-art approach to solve these problems is to combine propagation with conflict-driven search. This approach solved a number of open instances of the PSPLIB [KS96, PSP]. In this dissertation, we do not reevaluate the importance of a conflict relaxation for the



(a) Variables x_1, x_2, and x_3 fixed to one imply that variable x_4 can be fixed to zero.

(b) Variables x_1 and x_3 fixed to one imply that variable x_4 can be fixed to zero.

(c) Variables x_2 and x_3 fixed to one imply that variable x_4 can be fixed to zero.

Figure 2.1: This figure illustrates snapshots of an implication graph. It shows three possible implications for the inference discussed in Example 2.3. The sufficient condition of an implication is called reason (see Definition 2.4). In this case all three reasons are also explanation (see Definition 2.6).

resource-constrained project scheduling problem, we focus on the generalization of the known explanations which are needed for the conflict relaxation in the context of energy-based propagation algorithms.

In the following we recall the basis of the conflict analysis. In Section 2.2.2 we discuss the idea of conflict-driven search for variables with integer domains. Finally in Section 2.2.3 we present how the propagation algorithms introduced in Section 1.4.3 for the cumulative constraint contribute to the conflict analysis.

2.2.1 Background

We first discuss how conflict constraints are retrieved from infeasible sub-problems. Second, we summarize how conflict constraints can be used to drive search.

Conflict analysis

The task of propagation algorithms is to infer variable domain restrictions. Therefore, these algorithms typically take the current (local) variable domains and a subset of constraints as input and return a smaller domain space. Hence, these algorithms infer bound changes (see Definition 1.9). For a single bound change usually the inference can be made with only a subset of the variable domains.

Example 2.3. Given binary variables $x_0, \dots, x_4$ and a knapsack constraint

$$x_0 + x_1 + x_2 + 4\,x_3 + 6\,x_4 \leq 10. \tag{2.7}$$

If the variables x_1, x_2, and x_3 are fixed to one, the constraint implies that variable x_4 must be fixed to zero. This is the case since the first four variables consume at least $0 + 1 + 1 + 4 = 6$ units of the knapsack capacity, leaving only 4 units for the last variable which is less than the weight of x_4. If this constraint is part of a larger COP which contains more variables, the other variables are irrelevant for this bound change. In addition only a subset of the variables belonging to the scope of the constraint might be needed. For the inference in our example the bound change $[\![x_4 \leq 0]\!]$ is already implied by the knapsack constraint if variable x_3 and one of the two variables x_1 and x_2 are fixed to one.

Which variable domain bounds imply which bound change can be depicted in an acyclic directed graph in which each vertex corresponds to a bound change. Each directed arc (u, v)

in this graph states a necessary condition: the bound change u is necessary to infer bound change v. All incoming arcs of a vertex v form a sufficient condition for the corresponding bound change. Bound changes that were directly created by branching decisions have no incoming arcs. They are the sources of the directed graph. The sinks of this graph belong to the current (local) bound changes which have not been used yet to infer other bound changes. We call this graph the *implication graph*.[1] Figure 2.1 shows snapshots of an implication graph which illustrate three possible implications for the bound change discussed in Example 2.3. The sufficient condition of an implication is called *reason* since it justifies the implied bound change. Formally a reason is defined as:

Definition 2.4 (reason). Given a COP $= (X, \mathfrak{C}, \mathfrak{D}, f)$. A *reason* for a bound change B_i for variable x_i is a set of bound changes $\mathcal{B}$ such that these bound changes together with the COP imply the bound change B_i. That is

$$\bigcap_{\tilde{B} \in \mathcal{B}} \tilde{B} \cap X_{\text{COP}} \subseteq B_i.$$

This means that adding the bound changes $\mathcal{B}$ of a reason to the COP causes all remaining feasible solutions (if any) to lie in the halfspace defined by B_i.

Example 2.5 (continuing Example 2.3). Figure 2.1 shows three possible implications for the bound change discussed in Example 2.3. The three reasons are:

$$\begin{aligned}\mathcal{B}_a &= \{[\![x_1 \geq 1]\!], [\![x_2 \geq 1]\!], [\![x_3 \geq 1]\!]\}\\ \mathcal{B}_b &= \{[\![x_1 \geq 1]\!], [\![x_3 \geq 1]\!]\}\\ \mathcal{B}_c &= \{[\![x_2 \geq 1]\!], [\![x_3 \geq 1]\!]\}\end{aligned}$$

Each of these reasons together with the knapsack constraint (2.7) imply that the binary variable x_4 can be fixed to zero.

The reasons stated in Example 2.5 are called *explanations* since their bound changes respect the (local) variable domains. In contrast,

$$\mathcal{B} = \{[\![x_0 \geq 1]\!], [\![x_3 \geq 1]\!]\}$$

is also a reason for the bound change $[\![x_4 \leq 0]\!]$. However, since the domain of variable x_0 is not restricted in our example, this reason contains a bound change ($[\![x_0 \geq 1]\!]$) which is not present at the time of propagation.

Definition 2.6 (explanation). An *explanation* is a reason (see Definition 2.4) with the additional condition that the reason only uses bound changes which are not tighter than the bound changes which are present during the inference.

The trivial reason $\mathcal{B} = \{B_i\}$ for a bound change B_i is not an explanation since this single bound change B_i is tighter than the bound of the corresponding variable before the propagation.

To construct an implication graph, each bound change which is not a branching decision needs to be linked to an explanation. If a propagation algorithm detects an inconsistency,

[1]The term conflict graph is also often used [Ach07b, Ach07a].

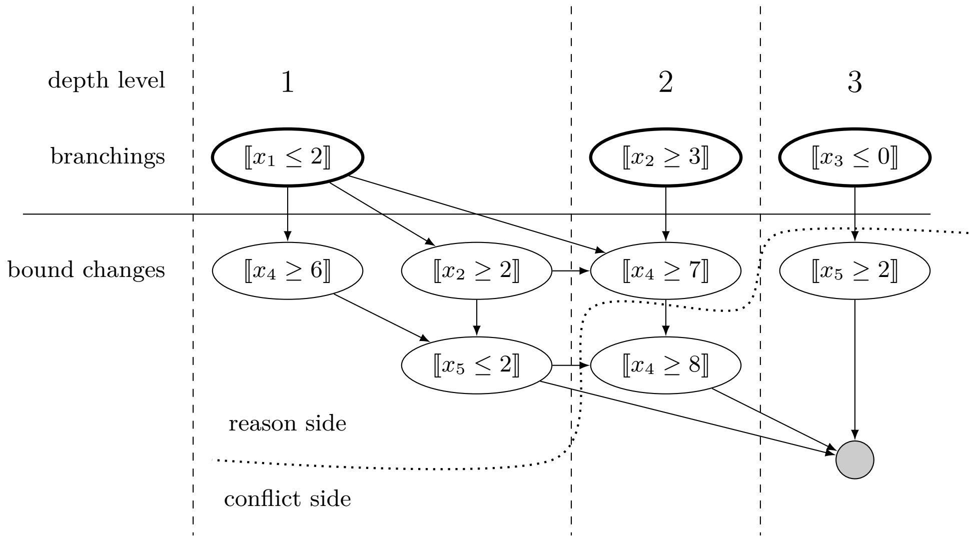


Figure 2.2: The figure illustrates an implication graph which could be at hand in case of an infeasible sub-problem. The dotted line indicates a conflict cut which separates the artificial infeasible vertex (●) from the vertices representing the branching decisions. The conflict cut splits the vertices in two partitions. The vertices belonging to the *conflict side* and the vertices belonging to the *reason side*. This conflict cut leads to the conflict $[\![x_5 \leq 2]\!] \wedge [\![x_4 \geq 7]\!] \wedge [\![x_3 \leq 0]\!]$.

an artificial vertex is added which represents the infeasibility. The explanation provided by the propagation algorithm forms a so-called *initial explanation* for the infeasibility. The corresponding arcs are added to the implication graph. In case of an infeasible sub-problem we have an implication graph at hand for which:

- ▷ each vertex (except the artificial infeasible vertex) represents a bound change,
- ▷ all sources belong to branching decisions,
- ▷ an artificial vertex exists which represents the infeasibility and is a sink in the directed graph, and
- ▷ all incoming arcs of each vertex represent an explanation for the corresponding bound change or infeasibility.

Example 2.5 already showed that explanations in general are not unique. Consequently, the implication graph is not unique.

In the case of an infeasible sub-problem an implication graph is the basis of the conflict analysis. Figure 2.2 illustrates an implication graph. Each cut separating the branching vertices (all sources of the graph) from the artificial infeasible vertex is called a conflict cut. Hence, a conflict cut is a subset of arcs and separates the graph into the *reason side* and the *conflict side*. The reason side is the one containing the branching decisions. The artificial infeasible vertex belongs to the conflict side. All bound changes belonging to the tail of the arcs of a conflict cut form a conflict with the property that if these bounds are added to the original problem the problem becomes infeasible. Therefore, a conflict constraint can be represented as an intersection of bound changes:

$$\bigcap_{x_i \in X_1} [\![x_i \geq a_i]\!] \cap \bigcap_{x_i \in X_2} [\![x_i \leq b_i]\!]$$

where X_1 and X_2 are subsets of the variable set X and it holds that adding these bound changes to the (original) problem renders it infeasible. Hence, at least one of these bound changes needs to be violated to achieve feasibility. Identical reasoning on a clause is used to justify *unit propagation* in SAT [ZS96].

Conflict analysis aims to find conflict constraints which provide additional restrictions to the search space. In practice the conflict cuts related to so-called *unique implication points* [MSS99, ZMMM01] of the different branching levels are often used. Formally, a unique implication point of a fixed branching level is a vertex in the implication graph which was inferred in this branching level or in a later branching level with the property that any path from the branching bound change to the artificial infeasible vertex needs to pass through this particular vertex. In general a branching level can have multiple unique implication points. The implication graph shown in Figure 2.2 has two unique implication points for the last branching level which are the vertices representing the bound changes $[\![x_3 \leq 0]\!]$ and $[\![x_5 \geq 2]\!]$. In practice we are interested in those that are closest to the vertex representing the infeasibility. These are called *first unique implication points.* For more details we refer to [MSS99, ZMMM01, Ach07b, Ach07a].

Summarizing, in case of an infeasible sub-problem, an implication graph is the input for the conflict analysis. This graph stores which bound changes imply which bound change. To be able to construct such a graph, each bound change needs to be linked to an explanation. An explanation is a set of bound changes with the property that adding these bound changes to the original problem restricts the remaining feasible solutions (if any) to the halfspace defined by the implied bound change (see Definition 2.6). In Section 2.2.3 we develop explanations for the inferences provided by the energy-based propagation algorithms introduced in Section 2.1.

Conflict-driven search (VSIDS)

The conflict constraints form a relaxation of the original problem that can be used to drive the search. Assuming all variables are binary, the basic idea of conflict-driven search is to select a variable which appears most frequently in conflict constraints. This means, we are counting the number of times a binary variable appears with its lower bound ($[\![x \geq 1]\!]$) and with its upper bound ($[\![x \leq 0]\!]$) in conflict constraints. This branching heuristic is known as VSIDS (Variable State Independent Decaying Sum) [MMZ+01]. These statistics are dynamically adapted such that newly detected conflicts have a larger impact compared to older conflict constraints. By choosing a binary variable which appears in many conflict constraints, the hope is that the conflict relaxation infers additional bound changes via unit propagation of the single conflict constraints. In the 0-branch (1-branch) all conflicts which contain the upper (lower) bound of the binary variable are redundant for the remaining search sub-tree since the branching bound change violates for these conflict constraints a bound change. Thereby the branching is done via variables which play a role in proving infeasibility. The hope is that this heuristic keeps the search tree small. The statistic used for this branching heuristic can be adjusted by also counting those bound changes which appear in intermediate conflicts during the conflict analysis (see [GN07]). For more details about conflict-driven search we refer to [MMZ+01, JPMSM08].

2.2.2 Adaptive conflict-driven search

In case of a binary variable, it is sufficient to count the number of lower and upper bounds appearing in conflict constraints since the lower and upper bound values are unique. In addition there is no choice in splitting the problem if a binary variable is selected. The problem is split into two sub-problems by fixing the binary variable to zero or to one.

However, for a variable with integer domain we need to decide how the variable domain is split to create sub-problems. To generalize the idea of conflict-driven search for binary variables to variables with integer domains we need to count how often a variable appears as a lower or upper bound in combination with a certain bound value. Hence, we are counting how often certain bound changes arise in conflict constraints. To adapt VSIDS we select a bound change which appears most frequently in conflict constraints. For this selection, we do not only discriminate between lower and upper bounds but also by the bound value. The statistic we need maps each variable and domain value to the number of times it appears as lower or upper bound in conflict constraints. We call this a *value based* variable statistic and use it to generalize conflict-driven search for variables with integer domains.

2.2.3 Energy-based explanations

In Section 2.2.1 we recalled the basic idea of the conflict analysis, which is used to exploit infeasible sub-problems. The input of the conflict analysis is an implication graph. To be able to construct this graph any inference needs to be linked to an explanation: a set of bound changes which implies the inference, see Definition 2.6. Note that it is not required that the explanation is the *actual* set of bound changes used during the propagation. It is sufficient to provide any set of bound changes which imply the inference. In this section we develop generic explanations for the inferences provided by any propagation algorithm belonging to the class of energy-based propagation algorithms (see Section 2.1). For the propagation algorithms introduced in Section 1.4.3, which belong to this class, we apply these generic explanations and show that the resulting explanations include known explanations for the corresponding propagation algorithm.

As presented above, the inference of energy-based propagation algorithms relies on volume arguments. For a fixed non-empty time interval $[a, b)$, a lower bound $\underline{e}_j(a, b)$ on the energy consumption of each job j within this interval is used to estimate the total energy consumption. See Section 2.1 for more details. An explanation is a reason with the additional restriction that only bound changes are used which are not tighter than the ones present during the propagation. However, we will show that it is sometimes possible to identify a valid explanation that contains weaker bound changes than the ones present during the propagation, strengthening the conflict constraint.

Before we present explanations, we analyze which bounds are needed to enforce a lower bound $\underline{e}_j(a, b)$ on the energy consumption of a fixed job j within a non-empty interval $[a, b)$.

Lemma 2.7. Given an interval $[a, b)$ with $a < b$, a job j with earliest start time est_j and latest start time lst_j, a resource demand r_j, and a processing time p_j, if

$$0 < \underline{e}_j(a, b) \leq e'_j(a, b) = \max\{0, \min\{b - a, p_j, \mathrm{ect}_j - a, b - \mathrm{lst}_j\}\} \cdot r_j$$

then the bounds

$$[\![S_j \geq a + \tfrac{\underline{e}_j(a,b)}{r_j} - p_j]\!] \cap [\![S_j \leq b - \tfrac{\underline{e}_j(a,b)}{r_j}]\!] \tag{2.8}$$

enforce that job j consumes at least $\underline{e}_j(a, b)$ units of energy within the interval $[a, b)$ and

$$[\![S_j \geq \mathrm{est}_j]\!] \cap [\![S_j \leq \mathrm{lst}_j]\!] \subseteq [\![S_j \geq a + \tfrac{\underline{e}_j(a,b)}{r_j} - p_j]\!] \cap [\![S_j \leq b - \tfrac{\underline{e}_j(a,b)}{r_j}]\!].$$

Proof. Since $0 < \underline{e}_j(a,b) \leq e'_j(a,b)$, it follows that $\min\{b-a, p_j, \mathrm{ect}_j - a, b - \mathrm{lst}_j\} > 0$. Therefore,

$$a + \frac{\underline{e}_j(a,b)}{r_j} - p_j \leq a + \frac{e'_j(a,b)}{r_j} - p_j \leq a + \frac{(\mathrm{ect}_j - a)\cdot r_j}{r_j} - p_j = \mathrm{ect}_j - p_j = \mathrm{est}_j$$

and

$$b - \frac{\underline{e}_j(a,b)}{r_j} \geq b - \frac{e'_j(a,b)}{r_j} \geq b - \frac{(b - \mathrm{lst}_j)\cdot r_j}{r_j} = \mathrm{lst}_j\,.$$

This implies, that

$$[\![S_j \geq \mathrm{est}_j]\!] \cap [\![S_j \leq \mathrm{lst}_j]\!] \subseteq [\![S_j \geq a + \frac{\underline{e}_j(a,b)}{r_j} - p_j]\!] \cap [\![S_j \leq b - \frac{\underline{e}_j(a,b)}{r_j}]\!].$$

Finally, we prove the claimed energy consumption of job j within the interval $[a,b)$. We perform a case distinction.

Case $\underline{e}_j(a,b) = (b-a)\cdot r_j$: Hence, $p_j \geq (b-a)$ and

$$[\![S_j \geq a + \frac{\underline{e}_j(a,b)}{r_j} - p_j]\!] \cap [\![S_j \leq b - \frac{\underline{e}_j(a,b)}{r_j}]\!] = [\![S_j \geq b - p_j]\!] \cap [\![S_j \leq a]\!].$$

Therefore, job j is definitely processed during the whole interval $[a,b)$. This implies that job j consumes $(b-a)\cdot r_j$ units of energy in the interval.

Case $\underline{e}_j(a,b) = p_j \cdot r_j$: Hence, $p_j \leq (b-a)$ and

$$[\![S_j \geq a + \frac{\underline{e}_j(a,b)}{r_j} - p_j]\!] \cap [\![S_j \leq b - \frac{\underline{e}_j(a,b)}{r_j}]\!] = [\![S_j \geq a]\!] \cap [\![S_j \leq b - p_j]\!].$$

Job j is processed completely within the interval $[a,b)$ requiring $p_j \cdot r_j$ units of energy.

Case $\underline{e}_j(a,b) < \min\{(b-a), p_j\}\cdot r_j$: Hence,

$$a + \frac{\underline{e}_j(a,b)}{r_j} - p_j \overset{\underline{e}_j(a,b) < p\cdot r_j}{\overset{\downarrow}{<}} a < a + \frac{\underline{e}_j(b-a)}{r_j} \overset{\underline{e}_j(a,b) < (b-a)\cdot r_j}{\overset{\downarrow}{<}} b$$

This implies, that if job j is scheduled at its earliest start time $\mathrm{est}_j = a + \frac{\underline{e}_j(a,b)}{r_j} - p_j$, then job j overlaps with the corresponding interval by $\frac{\underline{e}_j(a,b)}{r_j}$ units. The following inequality proves that the job j also overlaps by the same amount, if it is scheduled at its latest start time $\mathrm{lst}_j = b - \frac{\underline{e}_j(a,b)}{r_j}$.

$$b - \frac{\underline{e}_j(a,b)}{r_j} + p_j \overset{\underline{e}_j(a,b) < p\cdot r_j}{\overset{\downarrow}{>}} b > b - \frac{\underline{e}_j(b-a)}{r_j} \overset{\underline{e}_j(a,b) < (b-a)\cdot r_j}{\overset{\downarrow}{>}} a$$

Hence, job j overlaps with the interval $[a,b)$ at least with $\frac{\underline{e}_j(a,b)}{r_j}$ units and therefore requires at least $\underline{e}_j(a,b)$ units of energy.

This proves that in any case the claimed energy is consumed by job j. □

In the following we present explanations for the inconsistency check and the domain propagation. In addition we apply these generic explanations to the propagation algorithms introduced in Section 1.4.3 and show that the known explanations for the corresponding propagation algorithms can be understood as instantiations of the generic explanations.

Inconsistency check

If an energy-based propagation algorithm detects an inconsistency w.r.t. an interval $[a, b)$, the cumulative lower bounds $\underline{e}_j(a, b)$ of the energy consumption of each job j within the interval is larger than the available energy (see Inequality (2.2)). The following theorem characterizes explanations for this type of inference.

Theorem 2.8. Given a cumulative constraint $(\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}, C)$ and an interval $[a, b)$ with $a < b$, let us denote by $\mathcal{J}$ the set of jobs which need to be scheduled. Assume an inconsistency w.r.t. the energy-based propagation algorithm, that is $\sum_{j\in\mathcal{J}} \underline{e}_j(a, b) > (b - a) \cdot C$, where $\underline{e}_j(a, b)$ denotes the used lower bound on the energy consumption of job j within interval $[a, b)$. An explanation can be constructed from any subset $\Omega \subseteq \mathcal{J}$ which satisfies the necessary condition $\sum_{j\in\Omega} \underline{e}_j(a, b) > (b - a) \cdot C$. Then, the disjunction

$$\bigcap_{j\in\Omega \,:\, \underline{e}_j(a,b)>0} \Big([\![S_j \geq a + \tfrac{\underline{e}_j(a,b)}{r_j} - p_j]\!] \cap [\![S_j \leq b - \tfrac{\underline{e}_j(a,b)}{r_j}]\!] \Big). \tag{2.9}$$

is an explanation for the inconsistency in interval $[a, b)$.

Proof. Given are a cumulative constraint $(\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}, C)$ and a subset of jobs $\Omega \subseteq \mathcal{J}$ which satisfies the necessary condition $\sum_{j\in\Omega} \underline{e}_j(a, b) > (b - a) \cdot C$. Such a subset needs to exist because of the assumption that $\sum_{j\in\mathcal{J}} \underline{e}_j(a, b) > (b - a) \cdot C$. Hence, $\Omega = \mathcal{J}$ satisfies the necessary condition.

Lemma 2.7 proves that the lower and upper bound chosen in the Conflict (2.9) for a job j with $\underline{e}_j(a, b) > 0$ are not tighter as those present at the detection of the inconsistency and enforce that job j surely consumes at least $\underline{e}_j(a, b)$ units of energy within the interval $[a, b)$. Hence, Conflict (2.9) forces the cumulative constraint to be infeasible since Ω satisfies the necessary condition of the theorem. This proves that Conflict (2.9) is an explanation for the inconsistency in interval $[a, b)$. □

In case the interval $[a, b)$ which contains an inconsistency is known, an explanation can be constructed in $\mathcal{O}(n)$ where n denotes the number of jobs which are in the scope of the corresponding cumulative constraint. This assumes that the lower bound contribution for a fixed job can be computed in $\mathcal{O}(1)$ which is the case for $e'_j(a, b)$. A linear algorithm for constructing an explanation can iterate over all jobs and add jobs to Ω until the sum of lower bounds on the energy consumption is sufficient. Alternatively, one could first sort the jobs w.r.t. their energy consumption. Sorting potentially decreases the number of jobs needed for an explanation but increases the complexity to $\mathcal{O}(n \log n)$. The lower and upper bounds used for each job $j \in \Omega$ in Conflict (2.9) are the weakest possible bounds that imply that the corresponding job j requires at least $\underline{e}_j(a, b)$ units of energy within the interval $[a, b)$. These bounds may be looser than the bounds at the time that the inconsistency is detected. We refer to Section 2.2.4 for a discussion about which explanation to choose.

Domain propagation

Assume that an energy-based propagation algorithm infers a lower bound for the earliest start time est_j of job j w.r.t. a non-empty interval $[a, b)$. This implies that the following precondition must have been satisfied

$$\sum_{i\in\mathcal{J}\setminus\{j\}} \underline{e}_i(a, b) + r_j \cdot (\min\{b, \text{ect}_j\} - \max\{a, \text{est}_j\}) > (b - a) \cdot C \tag{2.10}$$

with $\min\{b, \mathrm{ect}_j\} - \max\{a, \mathrm{est}_j\} > 0$. That means, if job j was scheduled at its earliest start time, it would overlap with interval $[a, b)$ and this would lead to an overload in the considered interval. Consequently, the following lower bound can be deduced

$$b - \left\lfloor \frac{1}{r_j} \left((b-a) \cdot C - \sum_{i \in \mathcal{J} \setminus \{j\}} \underline{e}_i(a,b) \right) \right\rfloor \leq S_j. \tag{2.11}$$

See Section 2.1 for more details. An explanation for this inference needs to imply that it is infeasible to schedule job j at its earliest start time (Inequality (2.10)) and that enough energy is present such that the earliest start time of job j can be sufficiently bounded from below. That is:

$$\sum_{i \in \mathcal{J} \setminus \{j\}} \underline{e}_i(a,b) \geq (b-a) \cdot C - (b - \mathrm{est}'_j) \cdot r_j. \tag{2.12}$$

where est'_j denotes the inferred lower bound for the start time of job j. Generally speaking, to avoid the overload, job j needs to request less energy in the interval $[a, b)$ than if it were scheduled at its earliest start time. It holds that

$$(b - \mathrm{est}'_j) < \min\{b, \mathrm{ect}_j\} - \max\{a, \mathrm{est}_j\} \tag{2.13}$$

which implies that Inequality (2.11) is satisfied if Inequality (2.12) is satisfied. The following theorem describes explanations for this type of inference.

Theorem 2.9. Given a cumulative constraint $(\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}, C)$ and an interval $[a, b)$ with $a < b$, let us denote by $\mathcal{J}$ the set of jobs which need to be scheduled. Further, let est'_j be the lower bound of start time variable S_j of job j. Assume this bound has been deduced by an energy-based propagation algorithm using $\underline{e}_i(a, b)$ as lower bound on the energy consumption of job $i \in \mathcal{J}$ within interval $[a, b)$. Hence, Inequality (2.11) holds. An explanation for this inference can be constructed from any subset $\Omega \subseteq \mathcal{J} \setminus \{j\}$ which satisfies the necessary condition

$$\sum_{i \in \Omega} \underline{e}_i(a,b) \geq (b-a) \cdot C - (b - \mathrm{est}'_j) \cdot r_j \tag{2.14}$$

Let Ω be such a subset. Then the corresponding explanation has the form:

$$\bigcap_{i \in \Omega \,:\, \underline{e}_i(a,b) > 0} \left([\![S_i \geq a + \tfrac{\underline{e}_i(a,b)}{r_i} - p_i]\!] \cap [\![S_i \leq b - \tfrac{\underline{e}_i(a,b)}{r_i}]\!] \right) \cap [\![S_j \geq a + (b - \mathrm{est}'_j) - p_j + 1]\!]. \tag{2.15}$$

Proof. Given a cumulative constraint $(\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}, C)$, an interval $[a, b)$ with $a < b$, and a start time variable S_j with an earliest start time $\mathrm{est}'_j > \mathrm{est}_j$ that has been inferred by an energy based propagation algorithm via Inequality (2.11). Further, let $\Omega \subseteq \mathcal{J} \setminus \{j\}$ be a subset of jobs which satisfies the necessary Condition (2.14). Such a subset exists since the theorem assumes that the bound change was made by an energy-based propagation algorithm. Hence, $\Omega = \mathcal{J}$ satisfies the necessary condition.

From Lemma 2.7 follows that

$$\bigcap_{i \in \Omega \,:\, \underline{e}_i(a,b) > 0} \left([\![S_i \geq a + \tfrac{\underline{e}_i(a,b)}{r_i} - p_i]\!] \cap [\![S_i \leq b - \tfrac{\underline{e}_i(a,b)}{r_i}]\!] \right)$$

is an explanation which forces a load of at least $\sum_{i\in\Omega} \underline{e}_i(a,b)$ within in the interval $[a,b)$.

It remains to be shown that

$$[\![S_j \geq \text{est}_j]\!] \subseteq [\![S_j \geq a + (b - \text{est}'_j) - p_j + 1]\!]$$

and that it is infeasible to schedule job j at its earliest start time $\text{est}_j = a+(b-\text{est}'_j)-p_j+1$.

For job j:

$$\begin{aligned} a + (b - \text{est}'_j) - p_j + 1 &\overset{(2.13)}{<} a + \min\{b, \text{ect}_j\} - \max\{a, \text{est}_j\} - p_j + 1 \\ &\leq a + \text{ect}_j - a - p_j + 1 \\ &\leq \text{est}_j + 1. \end{aligned}$$

Since all values are integer numbers, it follows that $a + (b - \text{est}'_j) - p_j + 1 \leq \text{est}_j$. Hence,

$$[\![S_j \geq \text{est}_j]\!] \subseteq [\![S_j \geq a + (b - \text{est}'_j) - p_j + 1]\!],$$

which shows that the Disjunction (2.15) is an explanation.

Scheduling job j at its earliest start time as suggested by Explanation (2.15), which is $a + (b - \text{est}'_j) - p_j + 1$, implies that job j overlaps with interval $[a,b)$ with $(b - \text{est}'_j + 1)$ units. Hence,

$$\sum_{i\in\Omega} \underline{e}_i(a,b) + (b - \text{est}'_j + 1) \cdot r_j \overset{\substack{r_j > 0 \\ \downarrow}}{>} \sum_{i\in\Omega} \underline{e}_i(a,b) + (b - \text{est}'_j) \cdot r_j \overset{\substack{(2.14) \\ \downarrow}}{\geq} (b - a) \cdot C.$$

This proves an overload within interval $[a,b)$ if job j is scheduled at $a + (b - \text{est}'_j) - p_j + 1$.

Therefore, Explanation (2.15) is sufficient to prove the validity of the lower bound update made by the propagation algorithm. $\square$

If the interval $[a,b)$ that was responsible for the inference is known, an explanation can be constructed in linear time in the same fashion as for the inconsistency case. If the energy contributions of all jobs are first sorted to reduce the number of jobs being part of an explanation, the complexity increases to $\mathcal{O}(n \log n)$. The lower and upper bounds used for each job $j \in \Omega$ in Conflict (2.15) are the weakest possible bounds that imply that the corresponding job j requires at least $\underline{e}_j(a,b)$ units of energy within the interval $[a,b)$. These bounds may be looser than the bounds at the time that the inference took place. We refer to Section 2.2.4 for a discussion about which explanation to choose.

Applying the generic explanations

In Section 2.1 we showed that the propagation algorithm time-tabling, edge-finding, time-tabling edge-finding, and energetic reasoning belong to the class of energy-based propagation algorithms.

Time-tabling. For a fixed interval $[t, t+1)$ the following lower bound on the energy consumption for a job j is used:

$$\underline{e}_j(t, t+1) = \begin{cases} r_j & \text{if } \text{lst}_j \leq t < \text{ect}_j \\ 0 & \text{otherwise.} \end{cases}$$

Lemmas 2.10 and 2.11 state explanations for the inconsistency case and domain update case, respectively, of this propagation algorithm.

Lemma 2.10. Let us be given a cumulative constraint $(\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}, C)$ and denote with $\mathcal{J}$ the set of jobs which need to be scheduled. An explanation for an overload due to the core profile at time point t, that is $\Gamma_{\mathcal{J}}(t) > C$, can be constructed from any subset $\Omega \subseteq \mathcal{J}$ which satisfies the necessary condition $\Gamma_{\Omega}(t) > C$. Then, the disjunction:

$$\bigcap_{j \in \Omega \,:\, t \in \gamma_j} \Big([\![S_j \geq t - p_j + 1]\!] \cap [\![S_j \leq t]\!] \Big). \tag{2.16}$$

is an explanation.

Proof. Follows from Theorem 2.8 using for a fixed interval $[t, t+1)$ and the following lower bound on the energy contribution of job j within in the considered interval:

$$\underline{e}_j(t, t+1) = \begin{cases} r_j & \text{if } \operatorname{lst}_j \leq t < \operatorname{ect}_j \\ 0 & \text{otherwise.} \end{cases}$$

Using $a = t$ and $b = t + 1$ the explanation can be reformulated as:

$$\begin{aligned} &\bigcap_{j \in \Omega \,:\, t \in \gamma_j} \Big([\![S_j \geq t - p_j + 1]\!] \cap [\![S_j \leq t]\!] \Big) \\ = &\bigcap_{j \in \Omega \,:\, \underline{e}_j(a,b) > 0} \Big([\![S_j \geq a - p_j + \tfrac{r_j}{r_j}]\!] \cap [\![S_j \leq b - \tfrac{r_j}{r_j}]\!] \Big) \\ = &\bigcap_{j \in \Omega \,:\, \underline{e}_j(a,b) > 0} \Big([\![S_j \geq a - p_j + \tfrac{\underline{e}_j(a,b)}{r_j}]\!] \cap [\![S_j \leq b - \tfrac{\underline{e}_j(a.b)}{r_j}]\!] \Big) \end{aligned}$$

□

Lemma 2.11. Given a cumulative constraint $(\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}, C)$, let us denote with $\mathcal{J}$ the set of jobs which need to be scheduled. Further, let est'_j be the lower bound implied by Inequality (1.4) for start time variable S_j of job j. We define $t = \operatorname{est}'_j - 1$. If $\operatorname{est}_j < \operatorname{est}'_j < \operatorname{est}_j + p_j$, an explanation of this lower bound change can be constructed from any subset $\Omega \subseteq \mathcal{J} \setminus \{j\}$ which satisfies the necessary condition $\Gamma_{\Omega}(t) > C - r_j$. Then, the disjunction

$$\bigcap_{i \in \Omega \,:\, t \in \gamma_i} \Big([\![S_i \geq t - p_i + 1]\!] \cap [\![S_i \leq t]\!] \Big) \cap [\![S_j \geq \operatorname{est}'_j - p_j]\!]. \tag{2.17}$$

is an explanation.

Proof. Follows from Theorem 2.9 using

$$\underline{e}_j(t, t+1) = \begin{cases} r_j & \text{if } \operatorname{lst}_j \leq t < \operatorname{ect}_j \\ 0 & \text{otherwise} \end{cases}$$

for a fixed interval $[t, t+1)$. From $a = t$, $b = t + 1$, and $\operatorname{est}'_j = b$ it follows:

$$\operatorname{est}'_j - p_j = a - b + 1 + \operatorname{est}'_j - p_j = a + (b - \operatorname{est}'_j) - p_j + 1.$$

Hence, the explanation can be reformulated as:

$$\begin{aligned} &\bigcap_{i \in \Omega \,:\, t \in \gamma_i} \Big([\![S_i \geq t - p_i + 1]\!] \cap [\![S_i \leq t]\!] \Big) \cap [\![S_j \geq \operatorname{est}'_j - p_j]\!] \\ = &\bigcap_{j \in \Omega \,:\, \underline{e}_j(a,b) > 0} \Big([\![S_j \geq a - p_j + \tfrac{\underline{e}_j(a,b)}{r_j}]\!] \cap [\![S_j \leq b - \tfrac{\underline{e}_j(a.b)}{r_j}]\!] \Big) \cap [\![S_j \geq a + (b - \operatorname{est}'_j) - p_j + 1]\!]. \end{aligned}$$

□

The two lemmas formulate explanations for the time-tabling algorithm that are known from the literature [SFSW11]. The proofs, however, show that these explanations can be derived from the general concept of energy based propagation algorithms.

Edge-finding, edge-finding time-tabling, and energetic reasoning. The three propagation algorithms edge-finding, time-tabling edge-finding, and energetic reasoning differ in the used lower bound for the energy consumption of job j for a fixed interval. Since these lower bounds are in the interval $[0, e'_j(a,b)]$ for any job j, Theorem 2.8 and Theorem 2.9 provide explanations for the inconsistency check and the domain update, respectively. These explanations match previously published explanations for the respective algorithms [SFSW11, SFS13].

Optional jobs

The propagation algorithms used when there exist optional jobs rely on the propagation algorithms without optional jobs. In Section 1.4.4 we introduced two algorithms. One collects all jobs which are assigned to the cumulative resource (i.e., if the corresponding binary variable has been fixed to one). For these jobs, the standard propagation algorithms of the cumulative constraint are called. The second algorithm aims at proving that jobs which are not assigned to the cumulative resource yet, cannot be assigned without an inconsistency. In the following we discuss an explanation for the first algorithm.

If an inconsistency or a bound update is infered by one of the propagation algorithms discussed in Section 1.4.3 for the cumulative constraint without optional jobs, an explanation can be constructed by extending the explanation for the inference algorithm. To receive an explanation for the cumulative constraint with optional jobs we add all binary variables which are fixed to one. Hence, we add the lower bound change of the corresponding binary variables. This explanation can be strengthened by adding only those binary variables to the explanation for which the corresponding jobs are required for the explanation of the inference algorithm. Therefore, we can reuse the explanation for the individual propagation algorithms of cumulative constraints.

2.2.4 Discussion

We introduced generic explanations for propagation algorithms which belong to the class of energy-based propagation algorithms. These generic explanations capture known explanations for the time-tabling, edge-finding, time-tabling edge-finding, and energetic reasoning. For all algorithms belonging to the class of energy-based propagation algorithms the same algorithm can be used to construct explanations. In the following we briefly discuss:

▷ How can we diversify the set of explanations?

▷ Which explanation to choose?

How can we diversify the set of explanations?

As mentioned before, the explanation used for conflict analysis does not need to be the same as the reason used for propagation. Certain propagation algorithms use weaker lower bounds during propagation which results from the fact that, for those lower bounds, propagation algorithms are known which have a smaller worst case complexity. The complexity

job j	p_j	r_j	est_j	lct_j	$e_j(1,7)$	$e'_j(1,7)$
1	4	1	2	7	4	4
2	5	1	1	7	5	5
3	7	2	0	7	0	12
4	3	2	2	12	0	0

Table 2.1: The table gives for four jobs the processing time p_j, the resource demand r_j, the earliest start time est_j, the latest completion time lct_j, the energy used by the edge-finding propagation algorithm for the time interval $[1,7)$ denoted with $e_j(1,7)$, and the energy used by the energetic reasoning propagation algorithm for the same time interval $e'_j(1,7)$.

for constructing an explanation for an energy-based propagation algorithm, however, is independent of the lower bounds used to detect the inconsistency or domain propagation.

Example 2.12. Consider four jobs $\mathcal{J} = \{1,2,3,4\}$ with the setup given in Table 2.1, a cumulative capacity $C = 2$, and a time interval $[1,7)$. The edge-finding propagation algorithm (see Section 1.4.3) bounds the energy consumption of a job j for an interval $[a,b)$ from below by

$$e_j(a,b) = \begin{cases} p_j \cdot r_j & \text{if } [\text{est}_j, \text{lct}_j) \subseteq [a,b) \\ 0 & \text{otherwise.} \end{cases}$$

That gives the following lower bound on the total energy consumption:

$$E_{\mathcal{J}}(1,7) = \sum_{j \in \mathcal{J}} e_j(1,7) = 4 + 5 + 0 + 0 = 9.$$

Using Inequality (1.8) we can bound the start time variable S_4 of job 4 from below with 6 since

$$S_4 \geq b - \left\lfloor \frac{1}{r_j} \left((b-a) \cdot C - E_{\mathcal{J}}(a,b)\right) \right\rfloor = 7 - \left\lfloor \frac{1}{2} \left((7-1) \cdot 2 - 9\right) \right\rfloor = 6.$$

Theorem 2.9 implies that the subset of jobs $\Omega = \{1,2\}$ can be used to construct an explanation for the bound change $[\![S_4 \geq 6]\!]$ using the energy consumption bound $e_j(1,7)$:

$$[\![S_1 \geq 1]\!] \cap [\![S_1 \leq 3]\!] \cap [\![S_2 \geq 1]\!] \cap [\![S_2 \leq 2]\!] \cap [\![S_4 \geq 0]\!].$$

For this example this is the only explanation if we restrict ourselves to the energy consumption used by the edge-finding propagation algorithm. If we consider the energy used by the energetic reasoning propagation algorithm

$$e'_j(a,b) = \max\{0, \min\{b-a, p_j, \text{ect}_j - a, b - \text{lst}_j\}\} \cdot r_j,$$

we can construct an explanation from the subset $\Omega = \{3\}$

$$[\![S_3 \geq 0]\!] \cap [\![S_3 \leq 1]\!] \cap [\![S_4 \geq 0]\!].$$

This is also a valid explanation for the bound change $S_4 \geq 6$.

The necessary condition in Theorem 2.8 and Theorem 2.9 uses $\underline{e}_j(a,b)$ as a lower bound on the energy contribution for a job j within interval $[a,b)$. These lower bounds match the lower bounds used by the propagation algorithm during propagation. This restriction can be relaxed. In addition to the selection of a subset of jobs, the energy contribution can be selected. For each job j with $e'_j(a,b) > 0$, an energy contribution $\underline{e}_j$ can be chosen with $0 < \underline{e}_j \leq e'_j(a,b)$. If the selected energy contributions are still sufficient for the necessary condition, the theorems prove explanations for the inconsistency check and the bound change. This gives a much larger set of explanation to choose from.

Which explanation to choose?

The implication graph can be constructed on demand in a bottom-up manner as it is done in the constraint based solver SCIP. This means that the implication graph is freshly constructed starting from the artificial infeasible vertex and explanations for inferences are added in the reverse order they were made during the tree search. One advantage of this approach is that, at the moment an explanation for a certain inference is requested, parts of the implication graph are already known. Therefore, one may search for an explanation which is a minimal extension of the implication graph, in the sense that, the selected explanation adds as few new vertices to the implication graph as possible. In addition the lazy construction of the implication graph allows global information which was not present during the propagation to be taken into account. For our computational experiments in Chapter 4 and Chapter 5 we always construct explanations which yield a minimal extension.

Our generalized concept of explaining inference made by energy-based propagation algorithms leads to the following idea: when constructing an explanation, one might use different, potentially weaker lower bounds for the energy consumption of a job w.r.t. a time interval, than the lower bound which was used for the inference. Firstly, that increases the number of explanations to choose from. Secondly, it helps to find explanations which require a small number of jobs. Both might consecutively lead to a smaller search tree being generated and therefrom to an improved solver performance. However, preliminary computational studies [HS11, Sch12] with different explanations indicated that the impact for solving resource-constrained project scheduling problems is marginal. We do not present computational results on this topic as part of this dissertation, but see this as a potential direction of future research.

2.3 Linear relaxations of optional jobs

For the cumulative constraint as defined in Section 1.4.1 little is known about linear relaxations using the start time variables. In [HY02] valid inequalities are presented which are applicable on the boundary of the efficient horizon (see Definition 1.19). In this section we focus on the cumulative constraint with optional jobs as introduced in Section 1.4.4. We first recall in Section 2.3.1 known linear relaxations and put these relaxations in the context of existing propagation algorithms for the cumulative constraint. As a result we develop in Section 2.3.2 new relaxations which can be constructed from any energy-based propagation algorithm (see Section 2.1).

2.3.1 Background

In the literature there are two relaxations proposed for the cumulative constraint with optional jobs. Here we label them as the *single* relaxation [YAH10] and the *edge-finding* relaxation [Hoo04, Hoo05a, Hoo07]. For each `optcumulative` constraint, the former constructs a single linear constraint whereas the latter potentially adds several linear constraints to the linear relaxation. The basic idea of these relaxations is to analyze the aggregated energy of the jobs as compared to the available energy. The energy of a job j is the product of the processing time and the resource demand and needs to be considered if the corresponding binary decision variable x_j is one. If the cumulative energy of a set of jobs is larger than the available energy, not all jobs can be processed on a resource. This is the same reasoning the edge-finding propagation algorithm uses for cumulative constraints (see Section 1.4.3) and explains our chosen name. In the following we formally recall these two relaxations.

Single relaxation

For each `optcumulative` constraint a single knapsack constraint is constructed which bounds the total energy (demand times processing time) required by the potentially assigned jobs by the available energy [YAH10]. The available energy is given by the time interval ranging from the earliest start time to the latest completion time of all jobs times the resource capacity. Formally, for an `optcumulative` constraint the knapsack constraint:

$$\sum_{j \in \mathcal{J}} p_j r_j \, x_j \leq C \cdot (\max_{j \in \mathcal{J}}\{D_j\} - \min_{j \in \mathcal{J}}\{R_j\}) \tag{2.18}$$

is added to linear relaxation. Note that this linear constraint is redundant if the available energy is larger than the total energy required by all jobs. In that case, it does not forbid any subset of jobs to be placed on the resource.

Edge-finding relaxation

The edge-finding relaxation is inspired by the edge-finding propagation algorithm for the cumulative constraint. This propagator reasons about the required energy of the individual jobs. For a non-empty time interval $[a, b)$, the energy of all jobs which have to be processed in that interval is aggregated. If the resulting total energy is larger than the available energy an inconsistency is discovered (see Section 1.4.3 for more details). The same idea is used to detect sets of jobs which cannot be placed at the same time on a resource. The basic idea to construct a linear relaxation is to collect all sets of jobs for which the edge-finding propagation algorithm may yield an infeasibility. For these jobs, a linear knapsack constraint is added to the linear relaxation which ensures that this job combination is not assigned to the resource. Since jobs are only considered if they are definitely to be processed within a certain time interval, a time window $[a, b)$ is only of interest if a matches an earliest start time job and b a latest completion time of a job. This gives potentially $\mathcal{O}(|\mathcal{J}|^2)$ linear constraints per `optcumulative` constraint. The edge-finding relaxation is formulated as follows:

$$\sum_{j \in \mathcal{J}} e_j(a, b) \, x_j \leq C \cdot (b - a) \quad \forall (a, b) \in \{R_1, \ldots, R_n\} \times \{D_1, \ldots, D_n\} \, : \, a < b \tag{2.19}$$

$$e_j(a, b) = \begin{cases} p_j \cdot r_j & \text{if } [R_j, D_j) \subseteq [a, b) \\ 0 & \text{otherwise.} \end{cases} \tag{2.20}$$

To avoid of adding inequalities which are dominated by others for a single `optcumulative` constraint, in [Hoo04, Hoo05a, Hoo07] a cubic algorithm (w.r.t. the number of jobs) is presented which detects non-dominated inequalities for a resource. This reduces the number of added inequalities.

Note, that the single relaxation (see Constraint (2.18)) is one of the linear constraints in the edge-finding relaxation and if all jobs have the same time window the edge-finding relaxation is identical to the single relaxation.

2.3.2 Energy-based linear relaxations

The basic idea used to construct a linear relaxation via the edge-finding propagation algorithm can be generalized to be applicable for all propagation algorithms which belong to the class of energy-based propagation algorithm.

An energy-base propagation algorithm as defined in Section 2.1 uses a lower bound $\underline{e}_j(a,b)$ for the energy consumption of job j within the interval $[a,b)$ for each job j and fixed non-empty interval $[a,b)$. These lower bounds are used to detect inconsistencies when the sum of lower bounds is larger than the available energy:

$$\sum_{j\in\mathcal{J}} \underline{e}_j(a,b) > (b-a)\cdot C.$$

If this is the case, clearly not all jobs can be processed on the corresponding resource and a non-optional cumulative constraint would detect infeasibility. For a cumulative constraint with optional jobs, these circumstances can be used to construct a linear knapsack constraint which can be added the linear relaxation:

$$\sum_{j\in\mathcal{J}} \underline{e}_j(a,b)\, x_j \leq C\cdot(b-a). \tag{2.21}$$

As a result each energy-based propagation algorithm implies a linear relaxation for cumulative constraints with optional jobs:

$$\sum_{j\in\mathcal{J}} \underline{e}_j(a,b)\, x_j \leq C\cdot(b-a) \quad \forall (a,b)\in\{R_1,\ldots,R_n\}\times\{D_1,\ldots,D_n\}\,:\, a<b. \tag{2.22}$$

These linear relaxations can be used for Benders decomposition approaches where the subproblem contains an `optcumulative` constraint. This is the case for the allocation and scheduling problem considered in Chapter 5.

Not all of these linear constraints will be helpful. Some of them will be redundant and do not need to be added to the linear relaxations. A basic algorithm for constructing a linear relaxation is as follows. Run the corresponding consistency check of the energy-based propagation algorithm assuming that all jobs are assigned to the resource. If the algorithm detects an infeasibility (overload) w.r.t. to a time interval $[a,b)$, a linear knapsack constraint of the form of Inequalities 2.21 is added to the linear relaxation. Instead of stopping the consistency check, continue to detect further time intervals which are overloaded. This simple algorithm has the same worst case complexity as the corresponding algorithm of the energy-based propagation algorithm which checks for inconsistencies and avoids to adding obviously redundant linear constraints. It can be executed for example after the presolving phase to construct a linear relaxation of a cumulative constraints with optional jobs which can be added to a linear relaxation of a COP.

This concept allows us to develop a new linear relaxation for the cumulative constraint with optional jobs by using the the energetic reasoning propagation algorithm [BPN01]. In Section 2.1 we showed that this propagation algorithm is an energy-based propagation algorithm. It uses the following lower bound for each job j and interval $[a,b)$ to bound the energy consumption of job j within this interval

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

This leads to a linear relaxation for a cumulative constraint with optional jobs:

$$\sum_{j\in\mathcal{J}} e'_j(a,b)\,x_j \leq C\cdot(b-a) \quad \forall(a,b)\in\{R_1,\ldots,R_n\}\times\{D_1,\ldots,D_n\}\;:\;a<b. \tag{2.23}$$

In Chapter 5 we use this linear relaxation for the cumulative constraint with optional jobs within a logic-based Benders decomposition approach, a mixed-integer programming approach, and a constraint integer programming approach for a resource allocation and scheduling problem.

2.4 Summary

In this chapter we discussed relaxations for cumulative constraints as defined in Section 1.4. We focused on the conflict relaxation and the linear relaxation. For both relaxations we developed a general concept to construct explanations and linear constraints, respectively, which work for any propagation algorithm which belongs to the class of energy-based propagation algorithms. This class was defined in Section 2.1.

In Section 2.2.3 we developed general explanations which can be applied for any inference made by a propagation algorithm belonging to the class of energy-based propagation algorithms. These general explanations rely on the observation that all these algorithms use a lower bound on the energy consumption of a fixed job j within a fixed interval whereas the volume formulas used for inference are the same (see Section 2.1). For particular algorithms these general explanations are identical to previously published explanations. The added value of this chapter lies in the generalization to a whole class of propagation algorithms.

For the linear relaxation of cumulative constraints with optional jobs, we used the same observation as for the conflict relaxation and showed how to construct a linear relaxation from any propagation algorithm belonging to the class of energy-based propagation algorithms, see Section 2.3.2. This general concepts captures known linear relaxations for this structure. In addition we presented a new linear relaxation which is a corollary of our concept using the energetic reasoning propagation algorithm.

The concepts used for the conflict relaxation and for the linear relaxation are tightly related. They rely on the same observation that energy-based propagation algorithms use a lower bound on the energy consumption for fixed job j and a fixed interval. The different lower bounds and intervals result in different linear relaxations for cumulative constraints with optional jobs and different explanations for the inferences made by an energy-based propagation algorithm. This gives a variety of explanations and linear relaxations to choose from. In general it cannot be judged which of these is best w.r.t. solving particular instances as that depends on, among other things, the solving system where these techniques are integrated.

3 Presolving reductions and dual reductions for cumulative constraints

In this chapter we present one of the main contributions of this dissertation. Inspired by the importance of a presolving phase [AW13] in the context of mixed-integer programming, we develop several presolving steps for the cumulative constraint in the context of constraint programming.

Before search, the presolving phase attempts to make the subsequent tree search faster

▷ by using *primal heuristics* to find feasible solutions [Ber06, Ber14],

▷ by gathering structural information that can be used to guide the tree search [AR10, Sal14], and

▷ by reformulating the problem by detecting redundant constraints and fixing variables as well as strengthening variable bounds and constraints [Sav94].

Redundant constraints can improve the model due to a different propagation though these constraints. It is not clear if removing them completely from the model is helpful. However, knowing which constraints are redundant can be used to speed-up the search. For example these constraints do not need to be checked if they are satisfied by a potential solution. In the case of redundant linear constraints, it will speed-up the process of calculating the linear relaxation if these constraints are not added to it.

In case of the cumulative constraint, the worst case complexity of most propagation algorithms depends on the number of jobs assigned to the corresponding resource (see Section 1.4.3). One goal is to shrink this number during the presolving phase, potentially speeding the propagation algorithms later during the search and possibly leading to free variables that can be fixed globally. Besides reformulating the cumulative constraints, it is also of interest to use presolving to contribute to the sources of global information which may be used by the remaining constraints and propagation algorithms to lead to further model reductions and algorithmic efficiencies.

In the following, we present several presolving steps which decrease the number of jobs assigned to a resource and tighten the variable domains. We not only combine the global structures within a single cumulative constraint, but also present methods which reason about a set of cumulative constraints. Furthermore, we show how techniques which were previously developed for specific scheduling problems can be generalized using the concept of global structures, making these algorithms applicable for any COPs containing cumulative constraints.

Contribution. This chapter is dedicated to generic presolving techniques for the cumulative constraint and contains the following contributions. In Section 3.3 we adapt the notions of variable locks and variable bounds to the cumulative constraint. In addition, we generalize known problem specific presolving techniques for the cumulative constraint, thus, making them available within a general purpose solver through the use of globally available structures, e.g., the variable bound graph. In Section 3.4, we apply the concept of dual reductions described in Section 1.7 for generic COPs and present several dual reductions for the cumulative constraint. These reductions are applicable to a single cumulative constraint, but are also generalized to a set of cumulative constraints. Section 3.5 states presolving methods for the cumulative constraint with optional jobs. These methods rely on presolving techniques for the cumulative constraint without optional jobs.

Previously published. The results presented in this part are joint work with Jens Schulz and J. Christopher Beck. Parts of them were previously published in the following papers:

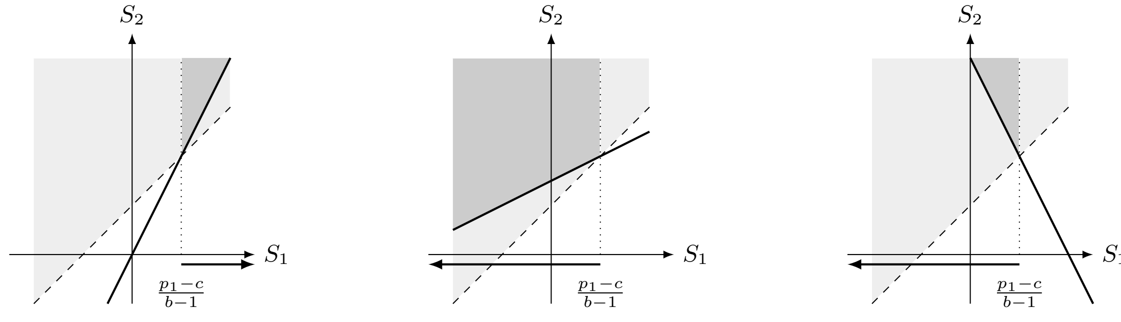
1. Stefan Heinz and J. Christopher Beck, *Reconsidering mixed integer programming and MIP-based hybrids for scheduling*, in Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2012), N. Beldiceanu, N. Jussien, and E. Pinson, eds., Lectures Notes in Computer Science 7298, Springer, 2012, pp. 211–227.

2. Stefan Heinz, Jens Schulz, and J. Christopher Beck, *Using dual presolving reductions to reformulate cumulative constraints*, Constraints 18, no. 2 (2013), pp. 166–201.

3. Stefan Heinz, Wen-Yang Ku and J. Christopher Beck, *Recent improvements using constraint integer programming for resource allocation and scheduling*, in Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, C. Gomes and M. Sellmann, eds., Lecture Notes in Computer Science, Springer, 2013.

Paper 2 generalizes the concept of dual reductions – which is well known for MIPs – to COPs. We applied this idea to the cumulative constraint and showed that for resource-constrained project scheduling problems, these reductions lead to a reduction of the model size (w.r.t. the number of non-fixed variables) after the presolving phase. This paper contains a subset of the techniques discussed in this chapter for the cumulative constraint. In Papers 1 and 3 we sketched presolving ideas for the cumulative constraint with optional jobs. The reformulated constraints tend to deliver a stronger linear relaxation.

Outline. This chapter is organized as follows: in Section 3.1 we introduce an algorithm which projects an arbitrary variable bound graph to a precedence graph. This precedence graph is needed as input for several presolving techniques presented later. A decomposition technique for splitting a single cumulative constraint into several cumulative constraints without losing any structural information is discussed in Section 3.2. The contribution of cumulative constraints to the global pools of variable locks and variable bounds is presented in Section 3.3. Additionally, we discuss the detection of disjunctive constraints (cumulative constraints with a resource capacity of one) and we generalize known problem specific presolving techniques for the cumulative constraints to be applicable in general purpose solver featuring a cumulative constraint. In Section 3.4 we present dual reduction techniques which are based on the variable locks and the variable bound graph. We first focus on single cumulative constraints and generalize these results to a set of cumulative constraints. Section 3.5 is dedicated to the cumulative constraint with optional jobs. We develop presolving methods for this type of constraint. We close this chapter by summarizing the presented presolving techniques in Section 3.6.



(a) $b > 1$ **(b)** $0 < b < 1$ **(c)** $b < 0$

Figure 3.1: Illustrating Lemma 3.2 and Lemma 3.3. The boundary of the precedence constraint $S_1 + p_1 \leq S_2$ is visualized by a dashed line (- - -). The border of the variable bounds condition $b \cdot S_1 + c \leq S_2$ is displayed by a solid line (——). The filled area (▨) shows the feasible region where the variable bound condition implies the precedence constraint which depends on the variable bound coefficient b.

3.1 Projecting variable bounds to the cumulative structure

Some of the presolving steps we discuss in the following require the knowledge of precedence conditions between jobs stating that a job can only start after another one is finished. Such a structure can be provided by the user or can be given implicitly via other constraints. In this section we discuss the use of the variable bound graph (see Section 1.3.1) to discover such precedence relationships between jobs. We use the variable bound graph to construct such a precedence graph.

In order to detect precedence conditions, we need a fixed processing time for each job which is associated with a start time variable. These processing times can be retrieved from the cumulative structure. Having these, we iterate through the arcs of the variable bound graph and check if a variable bound condition implies a precedence constraint of interest. Doing this for all variable bound conditions, we construct a precedence graph.

Consider a variable bound condition between two decision variable S_1 and S_2 of the form $b \cdot S_1 + c \leq S_2$. In case the variable bound coefficient b is one and the variable constant c is larger or equal to the processing time considered for S_1, we detect a precedence constraint. The following lemma formalizes this.

Lemma 3.1. Let S_1 and S_2 be the start time variables for two jobs and p_1 the processing time for the first one. If there exists a variable bound condition of the form $b \cdot S_1 + c \leq S_2$ with $b = 1$ and $c \geq p_1$, then the second job cannot start before the first job is finished. That is, $S_1 + p_1 \leq S_2$.

Proof. $S_1 + p_1 \overset{\overset{b=1}{\downarrow}}{=} b \cdot S_1 + p_1 \overset{\overset{p_1 \leq c}{\downarrow}}{\leq} b \cdot S_1 + c \leq S_2$. □

The following two lemmas justify the cases where the variable bound coefficient is not equal to one.

Lemma 3.2. Let S_1 and S_2 be the start time variables of two jobs and p_1 the processing time for the first one. If there exists a variable bound condition of the form $b \cdot S_1 + c \leq S_2$

```
Input: Variable bound graph $D^{\text{v}} = (\boldsymbol{S}, A^{\text{v}})$ with variable bound coefficient mapping
       $b : A^{\text{v}} \to \mathbb{R}$, variable bound constant mapping $c : A^{\text{v}} \to \mathbb{R}$, and a mapping
       $p : \boldsymbol{S} \to \mathbb{N}$ which assigns to each start time variable a processing time.
Output: Precedence graph $D^{\text{p}} = (\boldsymbol{S}, A^{\text{p}})$.
$A^{\text{p}} \leftarrow \varnothing$;
foreach $a \in A^{\text{v}}$ do
    $S^{\text{pred}} \leftarrow \text{tail}(a)$;
    if $b(a) = 1$ and $c(a) \geq p(S^{pred})$ then                              ◁ Lemma 3.1
        $A^{\text{p}} = A^{\text{p}} \cup \{a\}$;
    else if $b(a) > 1$ and $S^{pred} \geq \frac{p(S^{pred})-c(a)}{b(a)-1}$ then     ◁ Lemma 3.2
        $A^{\text{p}} = A^{\text{p}} \cup \{a\}$;
    else if $b(a) < 1$ and $S^{pred} \leq \frac{p(S^{pred})-c(a)}{b(a)-1}$ then     ◁ Lemma 3.3
        $A^{\text{p}} = A^{\text{p}} \cup \{a\}$;
```

Algorithm 1: Using the global source of variable bounds to construct a precedence graph. See Section 1.3.1 for the definition of the variable bound graph $D^{\text{v}} = (\boldsymbol{S}, A^{\text{v}})$.

with $b > 1$ and the start time variable S_1 is bounded from below by $\frac{p_1-c}{b-1}$, then job 2 does not start before the first job is finished. That is, $S_1 + p_1 \leq S_2$.

Proof. To prove the lemma we need to show that $S_1 + p_1 \leq S_2$ always holds. Figure 3.1(a) illustrates this setup.

$$\frac{p_1 - c}{b - 1} \leq S_1 \overset{(b-1)>0}{\overset{\downarrow}{\Leftrightarrow}} p_1 - c \leq (b-1)S_1 \Leftrightarrow S_1 + p_1 \leq b \cdot S_1 + c \leq S_2$$

□

Lemma 3.3. Let S_1 and S_2 be the start time variables for two jobs and p_1 the processing time for the first one. If there exists a variable bound condition of the form $b \cdot S_1 + c \leq S_2$ with $b < 1$ and the start time variable S_1 is bounded from above by $\frac{p_1-c}{b-1}$, then job 2 does not start before the first job is finished. That is, $S_1 + p_1 \leq S_2$.

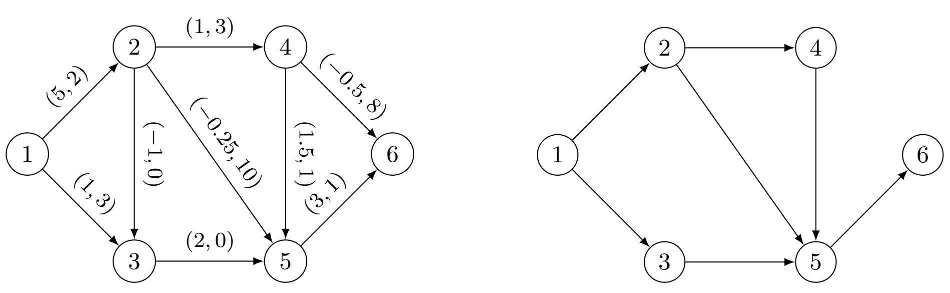
Proof. To prove the lemma we need to show that $S_1 + p_1 \leq S_2$ always holds. Figure 3.1(b) and 3.1(c) illustrate this setup for $0 < b < 1$ and $b < 0$, respectively.

$$\frac{p_1 - c}{b - 1} \geq S_1 \overset{(b-1)<0}{\overset{\downarrow}{\Leftrightarrow}} p_1 - c \leq (b-1)S_1 \Leftrightarrow S_1 + p_1 \leq b \cdot S_1 + c \leq S_2$$

□

Algorithm 1 summarizes the construction of a precedence graph using the availability of the variable bounds. Thereby, the variable bound graph $D^{\text{v}} = (\boldsymbol{S}, A^{\text{v}})$ (see Section 1.3.1) is scanned and a precedence graph $D^{\text{p}} = (\boldsymbol{S}, A^{\text{p}})$ is constructed. The following theorem formalizes conditions for a variable bound to imply a precedence constraint. We close this section with an example.

Theorem 3.4. Let S_1 and S_2 be the start time variables for two jobs and p_1 the processing time for the first one. A variable bound condition of the form $b \cdot S_1 + c \leq S_2$ implies the precedence condition with $S_1 + p_1 \leq S_2$ if one of the following conditions is satisfied:



(a) Variable bound graph **(b)** Resultant precedence graph

Figure 3.2: The figure displays a variable bound graph (a) and the implied precedence graph (b). Each arc of the variable bound graph is equipped with a tuple (b, c) stating the variable bound coefficient b and variable bound constant c (Example 3.6).

(i) $b = 1$ and $c \geq p_1$,

(ii) $b > 1$ and $S_1 \geq \frac{p_1 - c}{b-1}$, and

(iii) $b < 1$ and $S_1 \leq \frac{p_1 - c}{b-1}$.

Proof. Follows from Lemma 3.1, Lemma 3.2, and Lemma 3.3. □

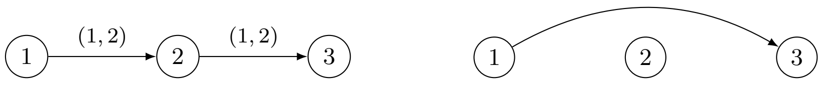
Remark 3.5. In Chapter 4 we consider resource-constrained project scheduling problems which contain a precedence graph. For these problems the variable bounds graph captures all precedence conditions (variable bounds with variable bound coefficient of one). For resource-constrained project scheduling problems without generalized precedence conditions, the precedence graph which results from Algorithm 1 includes the one given as input.

Example 3.6. Figure 3.2(a) depicts a variable bound graph for six variables. Each arc is equipped with a tuple (b, c) which represents the variable bound coefficient b and the variable bound constant c. For example, the arc between variable 1 and 2 represents a variable bound coefficient $b = 5$ and a variable bound constant $c = 2$. Hence, the variable bound reads $5 \cdot S_1 + 2 \leq S_2$. The variables have the following domains and corresponding processing times:

variable	1	2	3	4	5	6
lower bound	1	2	2	3	2	3
upper bound	5	5	8	12	7	8
processing time	3	2	1	2	4	1

Using Algorithm 1, the resulting precedence graph is shown in Figure 3.2(b). The arcs $(2, 3)$ and $(4, 6)$ cannot be mapped into a precedence condition.

Algorithm 1 tries to map a single variable bound condition to a precedence condition. The construction of a precedence graph, however, is not limited to this. Any source of globally valid information, which implies that a certain job needs to be finished before another can start, is usable. Figure 3.3 illustrates the detection of an additional precedence condition. Instead of trying to find an arc-to-arc mapping between the variable bound graph and



(a) Variable bound graph **(b)** Resulting precedence graph

Figure 3.3: Figure (a) visualizes a variable bound graph. The vertices display the variables and an arc is label with a tuple (b, c) giving the variable bound coefficient b and the variable bound constant c. If for all variables a processing time of 3 is assumed, the implied precedence graph is shown in Figure (b).

the precedence graph, a path-to-arc mapping can be considered. None of the arcs stated in Figure 3.3(a) imply a precedence condition if a processing time of 3 is assumed for all jobs. The path starting in vertex 1 and ending in vertex 3, however, implies a precedence condition between vertex 1 and 3. The exploitation of this observation is left for future work.

3.2 Decomposing a cumulative constraint

Figure 3.4 reprints a visualization of the effective horizon (see Definition 1.19) for Example 1.20. It shows a possibility of decomposing a cumulative constraint into two independent cumulative constraints. For the three jobs of Example 1.20, the unit capacity cannot be violated at time $t = 7$ since only job 3 can potentially be processed there. Hence, the resource with unit capacity can be modeled using two cumulative constraints with unit capacity where the first one contains jobs 1 and 3 and the second jobs 2 and 3. The following lemma formalizes this decomposition.

Lemma 3.7. Given a set of jobs $\mathcal{J}$, each $j \in \mathcal{J}$ with resource demands r_j and processing time p_j, that have to be scheduled on a resource with capacity C, if there exists a time point $t \in H$ within the effective horizon such that

$$\sum_{j \in \mathcal{J}} \mathbb{1}_{[\mathrm{est}_j, \mathrm{lct}_j)}(t)\, r_j \le C,$$

then this resource restriction is decomposable into two resource constraints by splitting the set of jobs into $\mathcal{J}_1 = \{j \in \mathcal{J} \,:\, \mathrm{est}_j \le t\}$ and $\mathcal{J}_2 = \{j \in \mathcal{J} \,:\, \mathrm{lct}_j \ge t\}$.

Proof. We are given a set of jobs $\mathcal{J}$ with resource demands r_j and processing times p_j which need to be placed on a resource with capacity C. Assume there exists a time point $t \in H$ which satisfies the condition of the lemma. Let $\mathcal{J}_1$ and $\mathcal{J}_2$ be the decomposition suggested by the lemma. To prove the lemma we show that the original constraint and both constraints resulting from the decomposition allow for the same set of feasible solutions.

First, let $\hat{\boldsymbol{S}}$ be a feasible solution for the original constraint. Since $\mathcal{J}_1$ and $\mathcal{J}_2$ are (proper) subsets of $\mathcal{J}$ and the two resource restrictions have the same capacity as the original one, it follows that $\hat{\boldsymbol{S}}$ is a feasible assignment for both constraints of the decomposition.

Second, let $\hat{\boldsymbol{S}}$ be a feasible assignment for both constraints formed by the decomposition. The resource restriction belonging to $\mathcal{J}_1$ has the same hmin as the original one and ensures that the resource capacity of the original constraint is not violated within the time window $[\mathrm{hmin}, t)$. Analogously the second condition related to $\mathcal{J}_2$ ensures that the capacity is not exceeded within the time window $[t, \mathrm{hmax})$. Since by definition $\mathcal{J}_1$ and $\mathcal{J}_2$ contain all

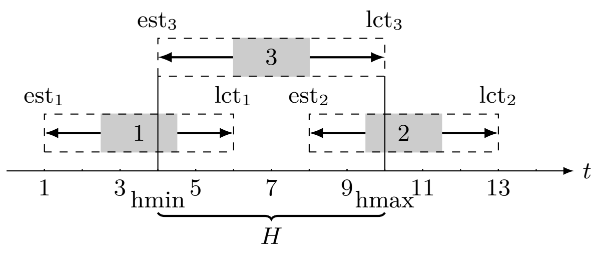


Figure 3.4: Reprinting Figure 1.4(b). Each shown job has demand of one. If a resource capacity of one is assumed the effective horizon is $H = [4, 10)$.

jobs which can potentially be processed within the time window $[\text{hmin}, t)$ and $[t, \text{hmax})$, respectively, it follows that $\hat{\boldsymbol{S}}$ is a feasible assignment for the original constraint as well.

□

Let us note that the sets $\mathcal{J}_1$ and $\mathcal{J}_2$ are not disjoint in general but $\mathcal{J}_1$ and $\mathcal{J}_2$ are both proper subset of $\mathcal{J}$ which is a result of the condition required for the time point t.

Remark 3.8. A cumulative constraint can be soundly decomposed in the same fashion as stated in Lemma 3.7 even if for the chosen time point, the condition of the lemma is not satisfied. In this case, the reformulations can lead to a weaker formulation since some of the structure is lost and inference might get weaker. This, however, is not the case if the condition regarding the chosen time point is satisfied.

3.3 Gathering structural information

Baptiste and Le Pape [BP00] analyzed resource-constrained project scheduling problems and discussed techniques to add redundant disjunctive constraints (cumulative constraints with capacity one) to strengthen the propagation during search. Thereby, they basically perform a presolving phase to collect useful information. To do this they did not only consider the structure of cumulative constraints, but also the existence of precedence constraints between jobs. These are special variable bounds where the variable bound coefficient is one (see Lemma 3.1). Schulz [Sch12] followed the idea of combining these two structures and presented methods to generated precedence constraints to improve the propagation in the subsequent tree search.

In Section 1.3, we discussed available sources of globally valid structures and indicated their usefulness for solving MIPs efficiently within a constraint-based system like SCIP. In this section, we show how the cumulative constraint contributes to these sources. Furthermore, we discuss a realization of the problem specific techniques presented in [BP00, Sch12] using the notion of global sources, making these methods available for any COPs containing cumulative constraints.

3.3.1 Exploiting variable locks

In Section 1.3.2 we discussed the notion of variable locks which have to be set by each constraint. These locks form a source of global information. In this section we explore the way a cumulative constraint contributes to this source.

For linear inequalities the variable locks are independent of the variable domain. They only depend on the sign of the coefficient (see Example 1.14). Therefore, the variable locks

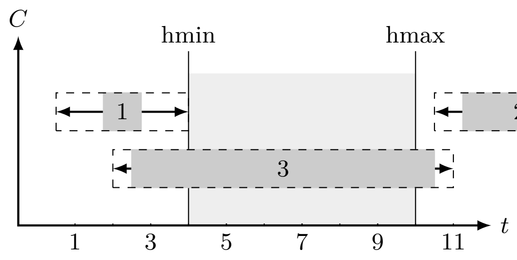


Figure 3.5: Illustration of Lemma 3.9 and Lemma 3.10. If the effective horizon is given by $[\text{hmin}, \text{hmax}) = [4, 10)$, all shown job are irrelevant, either because they do not overlap with the effective horizon (job 1 and 2) or because they are always processed throughout the complete effective horizon (job 3).

only change if a variable is removed from a linear constraint or the sign of the coefficient flips. This is not the case for the cumulative constraint. Here, the variable domains play an important role.

In general, a cumulative constraint must lock each start time variable in both directions since shifting a job in any direction may result in an infeasibility. In such a case, none of the dual reductions described in Section 1.3.2, can be applied for the variables belonging to a cumulative constraint since the locks are strictly greater than zero. We define and justify situations in which a cumulative constraint can omit variable locks for a start time variable since the constraint is monotonically decreasing or increasing in this start time variable (see Definition 1.11). We restrict ourselves to the down-locks. All results stated can be symmetrically transformed to the up-locks. For the up-locks we state a corollary and omit a proof.

We start by detecting *irrelevant* jobs ("completely free" start time variables). A job is called irrelevant for a cumulative resource (constraint) if an arbitrary assignment to its start time does not influence the assignment of any remaining jobs in that constraint. Such a job can be removed from the scope of the corresponding cumulative constraint. Definition 1.13 formalized this for an arbitrary constraint. Since the removed variable does not have to be locked by the constraint, the rest of the constraints in the problem gain dual information through the reduced number of locks. Figure 3.5 shows examples. The following two lemmas state this situation formally.

Lemma 3.9. Given a cumulative constraint $\mathcal{C} = (\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}, C)$, a start time variable S_j does not need to be locked in any direction if $\text{lct}_j \leq \text{hmin}$, $\text{est}_j \geq \text{hmax}$, $r_j = 0$, or $p_j = 0$.

Proof. The first two cases follow directly from the definition of hmin and hmax. In these two cases the job has no overlap with the effective horizon at all. Job 1 and 2 in Figure 3.5 illustrate this situation. The last two cases are obvious since the corresponding job requires zero energy. □

The previous lemma considers the situation where a job does not require any energy within the effective horizon. Now, we regard the case where a job must be processed throughout the effective horizon.

Lemma 3.10. Given a cumulative constraint $\mathcal{C} = (\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}, C)$, a start time variable S_j does not need to be locked in any direction if $\text{lst}_j \leq \text{hmin}$ and $\text{ect}_j \geq \text{hmax}$.

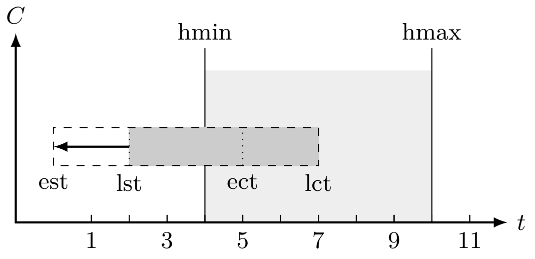


Figure 3.6: Illustration of Lemma 3.11. Shifting the corresponding job earlier would only relax the situation within the effective horizon $H = [\text{hmin}, \text{hmax})$ since the energy which needs be scheduled is reduced.

Proof. The conditions for the start time variable states that the corresponding job must start before or at the beginning of the effective horizon and must end after or at the end of the effective horizon. Job 3 in Figure 3.5 illustrates that situation. Independently of the actual start time of the job, it consumes its resource demand throughout the whole effective horizon. The start time assigned to the job does not impact the feasibility of any other variable assignments w.r.t. this cumulative constraint. Hence, no locking is required. □

Note that in both cases, the cumulative constraint is monotonically decreasing and increasing in variable S_j (see Definition 1.11). In case of Lemma 3.9 the corresponding start time variable can be removed from the scope of the cumulative constraint without any further adjustment. Due to the definition of the effective horizon, it follows that the corresponding job has a demand which is not larger than the capacity of the cumulative resource. For Lemma 3.10, however, the capacity of the corresponding cumulative constraint needs to be decreased by the demand of the removed job. Doing this might result in an infeasibility if the remaining demand is larger than the remaining capacity.

For jobs processed around hmin or hmax, we may be able to omit the down-lock or up-lock, respectively. Consider the situation that a job j has a latest start time $\text{lst}_j \leq \text{hmin}$. Depending on its processing time, this job could run either completely before the effective horizon or overlapping the effective horizon. In the latter case, we know that it overlaps with the effective horizon in a way such that hmin is included, see Figure 3.6 for an illustration. In this case moving the start time of this job earlier, out of the effective horizon H will always be feasible w.r.t. this constraint. Therefore, the cumulative constraint can omit the down-lock since it is monotonically decreasing w.r.t. that start time variable.

Lemma 3.11. Given a cumulative constraint $\mathcal{C} = (\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}, C)$, if $\text{lst}_j \leq \text{hmin}$, then the start time variable S_j does not need a down-lock w.r.t. $\mathcal{C}$.

Proof. We are given a cumulative constraint $\mathcal{C} = (\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}, C)$ and assume that, for start time variable S_j, the latest start time is $\text{lst}_j = D_j - p_j \leq \text{hmin}$. To prove that the cumulative constraint $\mathcal{C}$ does not need to down-lock S_j, we have to show that, for any two assignments $\boldsymbol{S}^1$ and $\boldsymbol{S}^2$ to the start time variables with $\boldsymbol{S}^1$ being a feasible assignment for $\mathcal{C}$, $S^1_i = S^2_i$ for $i \neq j$, and $S^1_j > S^2_j$, it follows that $\boldsymbol{S}^2$ is a feasible assignment (see Definition 1.12).

Per definition of the assignment for the start time variables, S^2_i is a feasible (partial) solution for the cumulative constraint $\mathcal{C}$ for all $i \neq j$. In case $S^2_j < \text{hmin} - p_j$ we know by the definition of hmin that $\boldsymbol{S}^2$ is a feasible assignment since job j is processed completely before the effective horizon. Thus, let us consider the remaining case $S^2_j \geq \text{hmin} - p_j$. Then, since $\text{lst}_j \leq \text{hmin}$, it follows that $S^2_j < S^1_j \leq \text{hmin}$. Hence, $\text{hmin} \leq S^2_j + p_j < S^1_j + p_j$.

Therefore. for all $t \in [\text{hmin}, \min\{S_j^2 + p_j, \text{hmax}\})$ we know that job j is also processed using the assignment $\boldsymbol{S}^1$. Hence, $\boldsymbol{S}^2$ is a feasible assignment.

□

Corollary 3.12. Given a cumulative constraint $\mathcal{C} = (\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}, C)$, if $\text{ect}_j \geq \text{hmax}$, then the start time variable S_j does not need a up-lock w.r.t. $\mathcal{C}$.

Example 3.13. If we recall Example 1.20 and assume that all three jobs have a processing time of 2 (see Figure 3.4), then for job 1 the down-lock and for job 2 the up-lock can be omitted. This follows, since $\text{lst}_1 = 4 = \text{hmin}$ and $\text{ect}_2 = 10 = \text{hmax}$.

3.3.2 Retrieving disjunctive constraints

Baptiste and Le Pape [BP00] presented a technique for resource-constrained project scheduling problems to detect redundant disjunctive constraints which potentially increase the propagation during the search. Using the concept of global sources and in particular the source of variable bounds (see Section 1.3.1), we sketch how this problem specific technique can be made available for any COP solver. Inspired by this idea we suggest a modified version to discover potentially useful disjunctive constraints.

The basic idea of Baptiste and Le Pape [BP00] is to create an incompatibility graph where each vertex represents a job and an edge is added if the corresponding two jobs cannot be processed in parallel. Baptiste and Le Pape [BP00] use three sources of global information to infer edges: the variable domains, the precedence constraints, which are naturally given for resource-constrained project scheduling problems, and the demands of the jobs w.r.t. available cumulative capacities. Each clique in an incompatibility graph corresponds to a set of jobs which cannot be processed in parallel. Hence, for such a set of jobs, a disjunctive constraint can be posted, stating that any two jobs are not processable in parallel. To avoid too many disjunctive constraints, Baptiste and Le Pape [BP00] search heuristically for a maximum clique. Further, they suggest to run this procedure multiple times, first considering the jobs for each individual cumulative constraint and then once taking all available cumulative constraints into account. Hence, at most the number of cumulative constraints plus one additional disjunctive constraints are created.

In Section 3.1, we discussed the construction of a precedence graph using the variable bound graph as input. Doing this allows us to apply the technique introduced by Baptiste and Le Pape [BP00] directly for any COPs containing cumulative constraints and variable bound structures. Basically, we project the variable bound structure into the cumulative structure and retrieve a precedence graph. This serves as input to discover disjunctive constraints.

We propose a slightly different way to detect disjunctive constraints by using the same three sources. Instead of searching for a disjunctive constraint for each cumulative constraint, we propose to search for a disjunctive constraint for each job. The idea is to construct for each job $j \in \mathcal{J}$ an incompatibility graph to find a disjunctive constraint which most likely infers bound changes for the corresponding start time variable S_j. To do that we assign to each node in the incompatibility graph, which represents a job, its processing time as weight, and search (heuristically) for a maximum weight clique [PX94]. While the maximum weighted clique problem is $\mathcal{NP}$-hard [GJ79], for small instances, as it is in our case, it is relatively quick to compute a good solution.

Let us be given a set of jobs $\mathcal{J}$, each job j with a processing time p_j, release date R_j, due date D_j, as well as a non-overlapping graph $G = (V, E)$[1] induced by the processing times and the cumulative structures (see Definition 1.22). Using these processing times and the variable bound structure, we build $D^{\mathrm{P}} = (\mathcal{J}, A^{\mathrm{P}})$[2], the transitive closure of the precedence graph, retrieved from Algorithm 1. For a fixed job j, we create an incompatibility graph $G^{\mathrm{C}} = (\mathcal{J}', E^{\mathrm{C}})$ with $\mathcal{J}' \subseteq \mathcal{J}$ as follows. The vertex set $\mathcal{J}'$ is given by:

$$\mathcal{J}' = \{j\} \cup \{i \in \mathcal{J} \setminus \{j\} \mid ji \in E\} \subseteq \mathcal{J}.$$

Hence, we only consider jobs that cannot overlap due to the non-overlapping graph G. This means that the sum of demands of job j and any job in $\mathcal{J}' \setminus \{j\}$ with respect to a cumulative condition is larger than the available capacity. We explicitly ignore those jobs which cannot be processed in parallel due to the precedence graph or the variable domains. These structures are already efficiently captured in the corresponding linear constraints and variable domains. Note that this selection is also applied in a similar fashion by Baptiste and Le Pape [BP00]. The edge set E^{C} for our construction is defined as:

$$E^{\mathrm{C}} = \{\{i,j\} \subseteq \mathcal{J}' \mid ij \in E \ \vee (i,j) \in A^{\mathrm{P}} \vee (j,i) \in A^{\mathrm{P}} \ \vee \mathrm{lct}_i \le \mathrm{est}_j \vee \mathrm{lct}_j \le \mathrm{est}_i\}.$$

We add an edge if an overlapping conflict exists w.r.t. one of the global sources. These are the non-overlapping structure of the cumulative constraints, the precedence structure implied by the variable bound graph, and the variable lower and upper bounds. Due to the construction of G^{C}, the vertex j has an edge to all other vertices. Therefore, j is always part of a maximum clique since its weight, given by the processing time of the corresponding job, is positive.

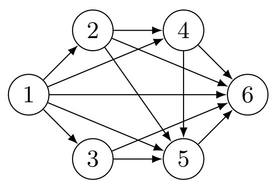
Remark 3.14. The construction of the edge set E^{C} is not limited to these three sources. Any information which is available and implies that two jobs cannot be processed in parallel is valid and can be used for expanding the edge set.

In a second step, we weight the vertices of G^{C}, each with the processing time of the corresponding job. The idea is find a set of conflict jobs with as much energy as possible in the potentially created disjunctive constraints where all jobs have a demand of one. Thereby, hoping that such a disjunctive constraint infers more. Using this weight function we compute a maximum weighted clique. This clique gives a set of start time variables which are pairwise in conflict with respect to being processed in parallel. This set is a candidate for creating a disjunctive constraint.

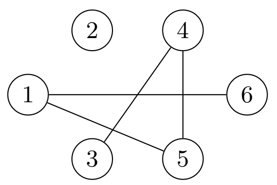
For all start time variables, we use this procedure to compute a set of jobs where each job pair cannot be processed in parallel. Before we create the disjunctive constraints, we filter duplicates and detect sets which are subset of others. Subsets can arise due to the fact that in the first step we restrict the vertices of the incompatibility graph depending on the chosen job j. The filtering can be realized in the same fashion as it is done for detecting duplicate SAT clauses [Blo70]. After the cleanup phase we are left with sets of jobs which have a non-empty relative complement in both directions. That is, if A and B are two of these sets with $A \neq B$ then $A \setminus B \neq \varnothing$ and $B \setminus A \neq \varnothing$. For these sets we create

[1]The non-overlapping graph $G = (V, E)$ is an undirected graph. The edge set E contains vertex sets of size two. For an edge of an undirected graph $\{i, j\} \in E$ we use the shorter notation $ij \in E$.

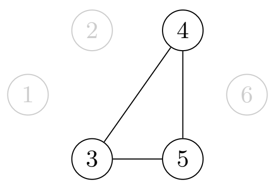
[2]The precedence graph $D^{\mathrm{P}} = (\mathcal{J}, A^{\mathrm{P}})$ is a directed graph. A^{P} is a set of ordered pairs of vertices. For an arc (directed edge) of an directed graph we use the notation $(i, j) \in A^{\mathrm{P}}$.



(a) Transitive closure of precedence graph



(b) Non-overlapping graph



(c) Incompatibility graph for variable 4

Figure 3.7: Illustration of Example 3.15. Figure (a) depicts the transitive closure of the precedence graph shown in Figure 3.2(b). Figure (b) displays a non-overlapping graph. For variable 4 the incompatibility graph is shown in Figure (c).

disjunctive constraints (cumulative constraint with unit demands and capacity) using the given processing times. Hence, each of these constraints is tailored to a particular job and we add at most as many disjunctive constraints as there are jobs.

Example 3.15 (Continuing Example 3.6). Example 3.6 illustrates the construction of a precedence graph using the global source of variable bounds and processing times. Continuing this example, we demonstrate the construction of an incompatibility graph for detecting disjunctive structures. Figure 3.7(a) shows the transitive closure of the precedence graph from Example 3.6. In addition Figure 3.7(b) presents a non-overlapping graph assuming the same processing time as for the construction of the precedence graph. Using these two graphs as input and choosing variable 4, the vertex set $\mathcal{J}'$ of the incompatibility graph contains the vertices, 3, 4, and 5. Figure 3.7(c) states the incompatibility graph. Hence for the clique $Q = \{3, 4, 5\}$ a disjunctive constraint can be added.

Remark 3.16. For disjunctive constraints this method can be used to lift jobs into the scope of a constraint. All jobs of a disjunctive constraint form a clique in the non-overlapping graph.

We close this section with a brief discussion of the usefulness of redundant disjunctive constraints. Most of the propagation algorithms for the cumulative constraint rely on energy arguments, i.e., they compare the available energy with the energy required by the jobs. For disjunctive constraints there is no remaining capacity when a job is processed. Therefore, the hope is that these constraints help to infer domain reductions early in the search.

3.3.3 Retrieving variable bounds

In the following, we analyze a possible contribution of the cumulative constraint to the variable bound graph which we introduced as a source of global information in Section 1.3.1. Thereby, we are not only using the cumulative constraints together with lower and upper bounds of the variables, as we did for the variable locks. In case of exploring variable bounds, we additionally consider the variable bounds which are already known, using the same three sources as in the previous section.

The idea is similar to the one discussed above. We create an incompatibility graph and search for a maximum (weighted) clique. This time the construction of the vertex set is customized to detect useful variable bounds. Let $\mathcal{J}$ be a set of jobs, each job j with a release date R_j, a due date D_j, and a processing time p_j. $G = (\mathcal{J}, E)$ is a non-overlapping graph which is induced by the cumulative constraints, and $D^{\text{P}} = (\mathcal{J}, A^{\text{P}})$ the transitive closure of the precedence graph resulting from the variable bound structure (see Section 3.1). The non-overlapping graph and the precedence graph depend on the given processing times. Instead of constructing an incompatibility graph for each job, we run this procedure for each arc of the transitive closure of the precedence graph (i.e., for each pair of jobs $(i, j) \in A^{\text{P}}$, for which a directed path from job i to job j in the implied precedence graph exists). We aim to compute a lower bound on the distance between the finishing time of job i and the start time of job j. This results in a variable bound conditions which can be added as a linear constraint to the model and into the global pool of variable bounds. The latter makes this information available for other algorithms.

The vertex set of the incompatibility graph $G^{\text{C}} = (\mathcal{J}', E^{\text{C}})$ for the pair of jobs $(i, j) \in A^{\text{P}}$ is a subset of $\mathcal{J}$ and defined as:

$$\mathcal{J}' = \{k \in \mathcal{J} \setminus \{i, j\} \mid [(i, k) \in A^{\text{P}} \wedge (k, j) \in A^{\text{P}}] \vee [\text{lct}_i \leq \text{est}_k \wedge \text{lct}_k \leq \text{est}_j]\}.$$

For constructing this vertex set we only consider the precedence conditions (implied by the variable bounds) and the variable domains. This set differs from that created in the previous section where only the cumulative structure was used. The idea is to collect all jobs which have to be processed between job i and job j. For finding such jobs we first use the transitive closure of the implied precedence graph and collect all jobs which are a direct successor of job i and a direct predecessor of job j. Second, we use the domains of the corresponding start time variables. Hence, each job $k \in \mathcal{J}'$ which can only start after job i is finished and has to be finished before job j (potentially) starts is added. Note that the vertex set can be empty ($\mathcal{J}' = \varnothing$).

Remark 3.17. The vertex set contains all jobs which are definitely processed between job i and job j. We consider the induced precedence graph and the domains of the start time variables to detect such jobs. Any other source which also implies that a particular job needs be processed between job i and job j is valid to expand this vertex set.

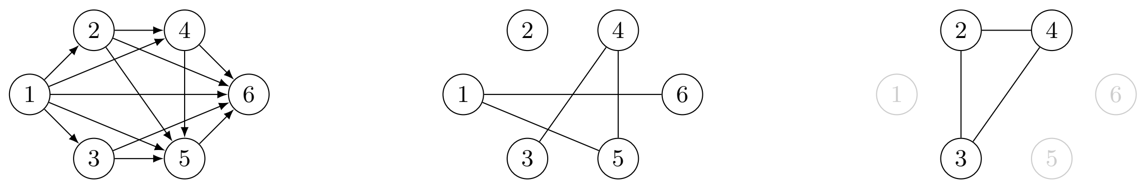
Edges are added between jobs which cannot be processed in parallel. This is done in the same fashion as in the previous section by utilizing the three sources of global information: two jobs are in conflict w.r.t. a cumulative constraint, i.e., the non-overlapping graph G contains an edge; a variable bound condition implies a precedence constraint; the feasible time windows do not overlap. Hence,

$$E^{\text{C}} = \{ij \subseteq \mathcal{J}' \mid ij \in E \ \vee (i, j) \in A^{\text{P}} \vee (j, i) \in A^{\text{P}} \ \vee \text{lct}_i \leq \text{est}_j \vee \text{lct}_j \leq \text{est}_i\}.$$

Again, the construction of the edge set is not limited to these three sources. Any other source implying that two jobs cannot run in parallel would be valid to use (see Remark 3.14).

Having this incompatibility graph $G^{\text{C}} = (\mathcal{J}', E^{\text{C}})$ for a pair of jobs (i, j), each clique $Q \subseteq \mathcal{J}'$ defines a set of jobs which need to be processed in sequence, between jobs i and j. Hence, the sum of processing times of jobs belonging to a clique, define a lower bound on the distance between the finish time of job i and the start time of job j. Formally we have

$$S_i + p_i + \sum_{k \in Q} p_k \leq S_j.$$



(a) Transitive closure of precedence graph

(b) Non-overlapping graph

(c) Incompatibility graph for variable pair $(1, 5)$

Figure 3.8: Illustration of Example 3.18. Figure (a) depicts the transitive closure of the precedence graph shown in Figure 3.2(b). Figure (b) displays a non-overlapping graph. For variable pair $(1, 5)$ the incompatibility graph is shown in Figure (c).

In order to find a clique which implies the largest lower bound, we weight each vertex with the processing time p_j of the corresponding job and search for a maximum weight clique.

Example 3.18 (Continuing Example 3.6). Considering the precedence graph implied by the variable bound structure of Example 3.6 and the non-overlapping graph used in Example 3.15, we construct the incompatibility graph for the variable pair $(1, 5)$. Figure 3.8(a) and Figure 3.8(b) state the transitive closure of the precedence graph and the non-overlapping graph, respectively. The vertex set of the incompatibility graph for the variable pair $(1, 5)$ is $\mathcal{J}' = \{2, 3, 4\}$. The corresponding jobs can only start after job 1 is finished and have to be completed before job 5 starts. Figure 3.8(c) illustrates the incompatibility graph. The clique $Q = \{2, 3, 4\}$ implies the following variable bound:

$$S_1 + 3 + \sum_{k \in Q} p_k \leq S_5 \Leftrightarrow S_1 + 8 \leq S_5.$$

Schulz [Sch12] used a similar idea in the context of resource-constrained project scheduling. He collects for a pair of jobs (i, j) all jobs which need to be processed between job i and job j using the precedence graph. For these jobs he proposes three different conditions to generate a lower bound on the time between the end job i and the start of job j. The first one uses a volume argument w.r.t. a particular cumulative capacity C. Another method computes a lower bound for a preemptive schedule of these jobs w.r.t. a particular cumulative condition. The last one uses a non-overlapping condition of two jobs w.r.t. the cumulative constraint, assumes that one job proceeds the other and vice versa, and computes for both assumptions the longest paths between job i and job j in the modified precedence graph. The minimum length of the two paths defines a lower bound as well. These three techniques can be applied for any COPs which contain cumulative constraints and variable bound structures by using the projection of the variable bounds presented in Section 3.1. Furthermore, the variable domains can be used to extend the set of jobs which need to be processed between job i and job j.

For resource-constrained project scheduling problems such redundant variable bound conditions can lead to an increase of the dual bound, i.e., the lower bound for the makespan. This might lead to an earlier optimality detection.

3.3.4 Strengthening variable bounds

In the previous section we discussed the detection of variable bound conditions. This section treats the strengthening of existing variable bounds using the cumulative condition.

A cumulative constraint implies a non-overlapping graph (see Definition 1.22). This graph captures the information that certain jobs cannot be processed in parallel since their cumulative resource demand is larger than the available capacity. We will show how this information can be used to strengthen existing variable bound conditions.

Lemma 3.19. Let S_1 and S_2 be the start time variables for two jobs and p_1 and p_2 the corresponding processing times. If the cumulative structure implies that the two jobs cannot be processed in parallel and there exists a variable bound condition of the form $S_1 + c \leq S_2$ with $-p_2 < c < p_1$, the variable bound condition can be strengthened to $S_1 + p_1 \leq S_2$.

Proof. If the variable bound condition $S_1 + c \leq S_2$ with $-p_2 < c < p_1$ is tight (meaning $S_1 + c = S_2$), it follows that job 1 and 2 overlap regardless of the actual start time assignment. This violates the cumulative structure since the jobs cannot be processed in parallel. Since $c > -p_2$, there is no chance that job 2 finishes before job 1 starts. Hence, job 2 can only start after job 1 is finished. That results in a stronger variable bound condition: $S_1 + p_1 \leq S_2$. □

For the classical resource-constrained project scheduling problem instances, containing only precedence conditions which state that certain jobs cannot be started before others are finished, this presolving technique does not apply. In case of the resource-constrained project scheduling problems with generalized precedence conditions, however, this technique can strengthen the model (see Chapter 4).

3.4 Dual reductions

In Section 3.3.1, we stated conditions under which the cumulative constraint is monotonically decreasing or increasing for a given start time variable (see Definition 1.11) and hence provides dual information via the variable locks. In this section, we use the knowledge of the variable locks within the cumulative constraint to infer dual reductions. We first present results for a single cumulative constraint and then generalize these to sets of cumulative constraints.

3.4.1 Single cumulative constraint

The variable locks define the number of constraints that "block" the shifting of a certain variable toward its lower or upper bound. A constraint can easily detect if it is the only one locking a certain variable by checking whether the corresponding locking number ζ^- or ζ^+ is one. Within the cumulative constraint, this information can be used to fix certain start time variables, based on a dual argument. If the cumulative constraint is the only one locking the start time variable down, the best bound of the variable w.r.t. the objective function is the lower bound (the objective function is monotonically non-decreasing in that variable), and fixing it to its lower bound results in a completion time before hmin, this variable can be dual fixed to its lower bound. Figure 3.9 illustrates this situation.

Lemma 3.20. Given a cumulative constraint $\mathcal{C} = (\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}, C)$, fixing a start time variable S_j with corresponding demand $r_j \leq C$ to its earliest start time (lower bound), that is $[\![S_j \leq \text{est}_j]\!]$, is dual feasible if the following conditions are satisfied:

(i) $\text{ect}_j \leq \text{hmin}$,

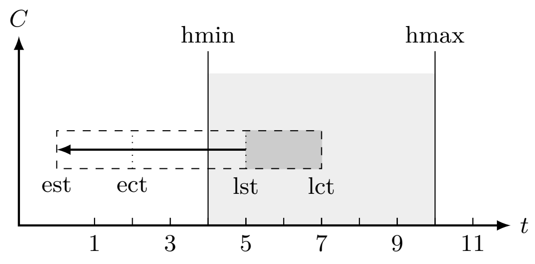


Figure 3.9: Illustration of Lemma 3.20. Fixing the start time variable to its earliest start time est results in a situation where this job does not influence the effective horizon H.

(ii) only the cumulative constraint $\mathcal{C}$ down-locks S_j, and

(iii) the objective function f is monotonically non-decreasing in S_j.

Proof. From the first condition, fixing the start time variable to its earliest start time means that the job finishes before hmin. Hence, by the definition of hmin, the assigned start time cannot lead to an infeasibility of the cumulative constraint.

By the definition of a variable lock, since only the cumulative constraint has down-locked this variable, none of the other constraints can be violated by fixing the start time variable to its earliest start time.

The third condition states that the objective function is monotonically non-decreasing in the start time variable S_j. Therefore, fixing it to the earliest start time (lower bound) is the best thing to do w.r.t. the objective function.

Hence, if the problem is feasible, there exists an optimal solution with job j starting at its earliest start time ($[\![S_j \leq \text{est}_j]\!]$). □

After fixing the variable to its earliest start time it follows that $\text{lct}_j \leq \text{hmin}$. Hence this job is irrelevant and can be removed from the constraint (see Lemma 3.9).

Corollary 3.21. Given a cumulative constraint $\mathcal{C} = (\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}, C)$, fixing a start time variable S_j with corresponding demand $r_j \leq C$ to its latest start time (upper bound), that is $[\![S_j \geq \text{lst}_j]\!]$, is dual feasible if the following conditions are satisfied:

(i) $\text{lst}_j \geq \text{hmax}$,

(ii) only the cumulative constraint $\mathcal{C}$ up-locks S_j, and

(iii) the objective function f is monotonically non-increasing in S_j.

Remark 3.22. Lemma 3.20 generalizes the "immediate scheduling rule" introduced by Baptiste and Le Pape [BP00] for resource-constrained project scheduling problems.

If the earliest start time of a job is smaller than hmin but the earliest completion is not, the previous lemma is not applicable even if the last two conditions hold. The following example illustrates such a situation.

Example 3.23. Given two jobs with unit demand and a resource with unit capacity. The first job has a release date of 1, a due date of 11, and a duration of 5. The second job is released at time 4, runs for 2 time steps, and has to be completion before time 7. Figure 3.10

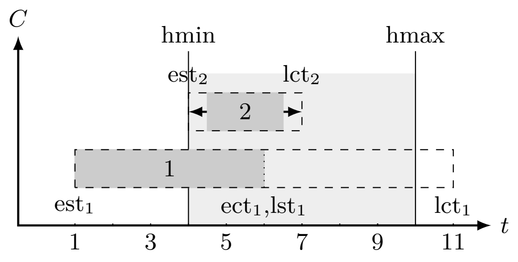


Figure 3.10: Illustrating job alignment of Example 3.23. Fixing job 1 to its earliest start time (lower bound) results in an infeasibility due to an overload at time point 5.

displays that situation. Fixing the first job to its earliest start time is not feasible since the second job cannot be processed anymore. However, fixing the second job to its earliest start time of 4 and the first job to it latest start time of 6 yields a feasible assignment for this resource.

In a setup as in the previous example, it is not possible to fix the start time variable of job 1 but it is dual feasible to remove some values from the domain of the start time variable. In the example, values 2, 3, and 4 can be removed from the domain of the start time variable belonging to job 1. Figure 3.11 depicts that statement and the following lemma formalizes it.

Lemma 3.24. Given a cumulative constraint $\mathcal{C} = (\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}, C)$ and a start time variable S_j, adding the constraint

$$[\![S_j \leq \text{est}_j]\!] \cup [\![S_j \geq \text{hmin} + 1]\!] \tag{3.1}$$

is dual feasible if the following conditions are satisfied:

(i) only the cumulative constraint $\mathcal{C}$ down-locks S_j, and

(ii) the objective function f is monotonically non-decreasing in S_j.

Proof. Given a cumulative constraint $\mathcal{C} = (\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}, C)$ and a start time variable S_j which satisfies the conditions of the lemma, if the proposed domain reduction does not remove any feasible solution of constraint $\mathcal{C}$, it is valid. Therefore, assume there exists a feasible assignment $\boldsymbol{S}^1$ for constraint $\mathcal{C}$ which violates Constraint (3.1), i.e., that $\text{est}_j < \text{hmin}$ and $S_j^1 \in \{\text{est}_j + 1, \ldots, \text{hmin}\}$. Due to Lemma 3.11 we know if $[\![S_j \leq \text{hmin}]\!]$ holds, the down-lock of S_j by the cumulative constraint $\mathcal{C}$ can be omitted. Hence, the assignment $\boldsymbol{S}^2$ with $S_i^2 = S_i^1$ for $i \neq j$ and $S_j^2 = \text{est}_j$ (shifting job j to its earliest start time), is feasible for constraint $\mathcal{C}$. Due to Condition (i), shifting the job earlier does not result in a violation of other constraints since all other constraints did not down-lock variable S_j. Condition (ii) ensures that this shift does not increase the objective value, meaning, $f(\boldsymbol{S}^2) \leq f(\boldsymbol{S}^1)$. Hence, the proposed domain reduction is dual feasible. $\square$

Note that in case of $\text{est}_j \geq \text{hmin}$, Constraint (3.1) is redundant. Hence, only for a start time variable with a lower bound smaller than hmin, can a dual reduction be deduced. In this case, the Constraint (3.1) defines a domain hole for the start time variable.

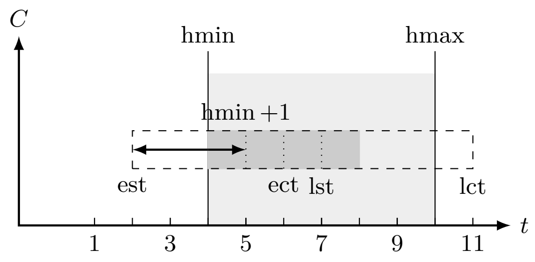


Figure 3.11: Illustration of Lemma 3.24. Fixing the start time variable to a value greater than the earliest start time and smaller than or equal to hmin is dual dominated by fixing the start time variable to its earliest start time.

Corollary 3.25. Given a cumulative constraint $\mathcal{C} = (\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}, C)$ and a start time variable S_j, adding the constraint

$$[\![S_j \geq \text{lst}_j]\!] \cup [\![S_j \leq \text{hmax} - p_j - 1]\!]$$

is dual feasible if the following conditions are satisfied:

(i) only the cumulative constraint $\mathcal{C}$ up-locks S_j, and

(ii) the objective function f is monotonically non-increasing in S_j.

Due to the special structure of resource-constrained project scheduling problem (without generalized precedence constraints and with all jobs having the same release date) it is known that for an optimal schedule at any point in time at least one job is processed. If this is not be the case, the schedule cannot be optimal since shifting all jobs, that are scheduled directly after the time point where no job is running, by one to an earlier start time results in a feasible solution which has a smaller makespan. This is possible since all resources have a constant capacity and only precedence conditions are given which state that certain jobs can only be started after others are finished. This observation lead to the branching idea *schedule-or-postpone* [PCVG94, BPN01]. The idea is to fix a job to its earliest start time (lower bound) and search for a feasible solution. After backtracking to the level where this decision was taken, the job is ignored (postponed) until its lower bound is updated via constraint propagation. By using the variable locks, it is possible to discover schedule-or-postpone situations where the minimum time by which a job can be postponed if it is not schedule at its earlier start time can be precomputed. Again, this is applicable within an arbitrary COP.

Theorem 3.26. Given a cumulative constraint $\mathcal{C} = (\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}, C)$ and a start time variable S_j, let us denote with $\mathcal{J}$ the set of jobs which need to be scheduled and define $\text{ect} = \min_{i \in \mathcal{J} \setminus \{j\}} \text{ect}_i$. Adding the constraint

$$[\![S_j \leq \text{est}_j]\!] \cup [\![S_j \geq \text{ect}]\!] \tag{3.2}$$

is dual feasible if the following conditions are satisfied:

(i) only the cumulative constraint $\mathcal{C}$ down-locks S_j and

(ii) the objective function f is monotonically non-decreasing in S_j.

Proof. Given a cumulative constraint $\mathcal{C} = (\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}, C)$ and a start time variable S_j which satisfies the conditions of the theorem, we assume for job j that $\text{est}_j < \text{ect}$, otherwise, nothing has to be shown since Constraint (3.2) would be redundant.

Let $\boldsymbol{S}^1$ be a feasible solution which violates Constraint (3.2). This implies

$$\text{est}_j < S_j^1 < \text{ect} = \min_{i \in \mathcal{J} \setminus \{j\}} \text{ect}_i \,.$$

To prove the theorem we have to show that $\boldsymbol{S}^2$ with $S_i^2 = S_i^1$ for all $i \in \mathcal{J} \setminus \{j\}$ and $S_j^2 = \text{est}_j$ is a feasible solution with $f(\boldsymbol{S}^2) \leq f(\boldsymbol{S}^1)$. The latter follows directly from Condition (ii), stating that the objective function is monotonically non-decreasing in start time variable S_j. Condition (i) ensure that shifting the start time variable S_j to an earlier start time does not affect the feasibility of the remaining constraints. Therefore, it remains to prove that $\boldsymbol{S}^2$ is a feasible assignment for the cumulative constraint $\mathcal{C}$. From the observation

$$\text{est}_j = S_j^2 < S_j^1 < \text{ect} \quad \text{with} \quad \text{ect} = \min_{i \in \mathcal{J} \setminus \{j\}} \text{ect}_i$$

it follows that for all jobs $i \in \mathcal{J} \setminus \{j\}$ the earliest completion time ect_i is strictly larger than the start time of job j w.r.t. $\boldsymbol{S}^1$. This means that none of these jobs is completed before job j starts. Hence, there is no job running at any time point $t < S_j^1$ which does not run at time point S_j^1, more precisely:

$$\mathbb{1}_{[S_i^1, S_i^1 + p_i)}(t) \leq \mathbb{1}_{[S_i^1, S_i^1 + p_i)}(S_j^1) \qquad \forall i \in \mathcal{J} \setminus \{j\},\ t < S_j^1$$

Since $\mathbb{1}_{[S_j^1, S_j^1 + p_j)}(t) = 0$ for $t < S_j^1$ and $\mathbb{1}_{[S_j^1, S_j^1 + p_j)}(S_j^1) = 1$ it follows that

$$\sum_{i \in \mathcal{J}} \mathbb{1}_{[S_i^1, S_i^1 + p_i)}(t)\, r_i + r_j \leq \sum_{i \in \mathcal{J}} \mathbb{1}_{[S_i^1, S_i^1 + p_i)}(S_j^1)\, r_i \leq C \qquad \forall t < S_j^1$$

Therefore, the load at time point $t < S_j^1$ is at least r_j smaller than the load at start point S_j^1 of job j. This allows job j to be moved to any earlier start time without violating constraint $\mathcal{C}$. Hence, $\boldsymbol{S}^2$ is a feasible assignment for the cumulative constraint $\mathcal{C}$ which proves the theorem. □

Note that if $\text{est}_j \geq \text{hmax}$, job j can be removed from the cumulative constraint (see Lemma 3.9). If S_j with $\text{est}_j \geq \min_{i \in \mathcal{J} \setminus \{j\}} \text{ect}_i$, Constraint (3.2) is redundant.

Corollary 3.27. Given a cumulative constraint $\mathcal{C} = (\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}, C)$ and a start time variable S_j, let us denote with $\mathcal{J}$ the set of jobs which need to be scheduled and define $\text{lst} = \max_{i \in \mathcal{J} \setminus \{j\}} \text{lst}_i$. Adding the constraint

$$[\![S_j \geq \text{lst}_j]\!] \cup [\![S_j \leq \text{lst} - p_j]\!]$$

is dual feasible if the following conditions are satisfied:

(i) only the cumulative constraint $\mathcal{C}$ up-locks S_j and

(ii) the objective function f is monotonically non-increasing in S_j.

Example 3.28 (continuing Example 3.23). As we observed before, Lemma 3.24 states that it is dual feasible to remove the values 2, 3, and 4 from the domain of the start time variable belonging to job 1. That means that job 1 starts at its earliest start time of 1 or not before time point $5 = \text{hmin} + 1$. Theorem 3.26 additionally eliminates the value 5 from the domain of the start time variable since

$$\text{ect} = \min_{i \in \mathcal{J} \setminus \{j\}} \text{ect}_i = \text{ect}_2 = 6.$$

For this setup, Theorem 3.26 results in more domain reductions than Lemma 3.24. Furthermore, the lemma and the theorem prove that job 2 can be fixed to its earliest start time of 4. Hence the only feasible assignment is to start job 2 at its earliest start time and job 1 at its latest start time.

Remark 3.29. Theorem 3.26 is not a generalization of Lemma 3.24 since in general for a given job j it does not hold that $\text{hmin} < \min_{i \in \mathcal{J} \setminus \{j\}} \text{ect}_i$. Theorem 3.26 does not consider the available capacity of the cumulative constraint, whereas Lemma 3.24 does via the usage of hmin. This observation also holds for a disjunctive constraint, where the capacity is fixed to one.

So far we have only considered variable locks as a source to infer dual reductions. If we additionally consider the variable bounds, we can find further reductions. Therefore, we follow the same idea as before. We create a precedence graph using the global variable bound structure as input for Algorithm 1. For the resource-constrained project scheduling problems we observed that for an optimal schedule, at any point in time at least one job is processed. This observation can be made applicable for general COPs containing a cumulative structure. It follows that at least one of the source jobs in the precedence graph (meaning it has no predecessor) has to start at its earliest start time (lower bound). The following theorem formalizes this dual reduction for any COPs.

Theorem 3.30. Given a cumulative constraint $\mathcal{C} = (\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}, C)$ and a precedence graph $D^{\text{P}} = (\mathcal{J}, A^{\text{P}})$ where $\mathcal{J}$ denotes the set of jobs which need to be scheduled, let $\mathcal{J}' \subseteq \mathcal{J}$ and $\text{ect} = \max_{i \in \mathcal{J}'} \text{ect}_i$. Adding the constraint

$$\bigcup_{j \in \mathcal{J}'} [\![S_j \leq \text{est}_j]\!] \tag{3.3}$$

is dual feasible if the following conditions are satisfied:

(i) $\forall j \in \mathcal{J} \setminus \mathcal{J}' \; \text{ect} \leq \max\{\text{hmin}, \text{est}_j\}$ or $\exists i \in \mathcal{J}'$ with $(i, j) \in A^{\text{P}}$,

(ii) only the cumulative constraint $\mathcal{C}$ down-locks the start time variables S_j with $j \in \mathcal{J}'$, and

(iii) the objective function f is monotonically non-decreasing for all start time variables belonging to the jobs in $\mathcal{J}'$.

Proof. Given a cumulative constraint $\mathcal{C} = (\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}, C)$, we denote with $\mathcal{J}$ the set of jobs which need to be scheduled. Let $\mathcal{J}' \subseteq \mathcal{J}$ be a subset which satisfies the assumptions of the theorem. In case $\text{ect} \leq \text{hmin}$ the claim follows directly from Lemma 3.20 since all jobs belonging to $\mathcal{J}'$ can be processed before the effective horizon starts. Therefore, we assume ect > hmin. Assumption (ii) ensures that "shifting" any job belonging to $\mathcal{J}'$ to the left (an earlier start time) does not introduce any infeasibility w.r.t. the remaining constraints. W.r.t. the optimality, Assumption (iii) safeguards that the assignment to an earlier start time does not worsen the objective value. Let us be given a feasible assignment $\boldsymbol{S}^1$ for $\mathcal{C}$ which violates Condition (3.3), i.e., none of the jobs belonging to $\mathcal{J}'$ is assigned to its earliest start time. Let $j \in \mathcal{J}'$ be a job which has the smallest completion time w.r.t. the assignment $\boldsymbol{S}^1$. That is, $j \in \operatorname{argmin}_{i \in \mathcal{J}'}(S^1_i + p_i)$. Showing that the assignment $\boldsymbol{S}^2$ with $S^2_i = S_i$ for all $i \in \mathcal{J} \setminus \{j\}$ and $S_j = \mathrm{est}_j$ is feasible proves the theorem.

The jobs $\mathcal{J} \setminus \mathcal{J}'$ can be partitioned into two sets:

$$\mathcal{J}^1 = \{i \in \mathcal{J} \setminus \mathcal{J}' \mid \mathrm{ect} \leq \mathrm{est}_i\}$$
$$\mathcal{J}^2 = \{i \in \mathcal{J} \setminus \mathcal{J}' \mid \exists k \in \mathcal{J}' : (k, i) \in A^{\mathrm{p}}\}.$$

These sets are not disjoint in general. Due to the selection of job j, we know that all jobs belonging to $\mathcal{J}^2$ do not start before job j finishes in the assignment $\boldsymbol{S}^1$. This is enforced by the existence of a precedence condition and the fact that job j has the smallest completion time. Assigning job j to an earlier start time does not change this. All jobs belonging to $\mathcal{J}^1$ start after the earliest completion time of job j. Hence starting job j at its earliest start time does not conflict with these jobs at all. It remains to show that moving job j to its earlier start time does not affect the jobs $\mathcal{J}' \setminus \{j\}$. We use the same argumentation as in the proof of Theorem 3.26. Since job j has by definition the smallest completion time within $\mathcal{J}'$, none of the jobs in $\mathcal{J}' \setminus \{j\}$ are completed before $S^1_j + p_j$ in assignment $\boldsymbol{S}^1$. Hence, for all $t < S^1_j$ it holds:

$$\sum_{i \in \mathcal{J}} \mathbb{1}_{[S^1_i, S^1_i + p_i)}(t)\, r_i + r_j \leq \sum_{j \in \mathcal{J}} \mathbb{1}_{[S^1_i, S^1_i + p_i)}(S^1_j)\, r_i \leq C.$$

This proves that during the time window $[\mathrm{est}_j, S^1_j)$ at least r_j resource capacity is available. Thus, assigning job j to its earliest start time does not conflict with $\mathcal{J}'$. Therefore, $\boldsymbol{S}^2$ is a feasible assignment for constraint $\mathcal{C}$ which proves the theorem. $\square$

The theorem proves that at least one of the jobs belonging to $\mathcal{J}'$ can be scheduled at its current earliest start time (lower bound). Thereby, the Conditions (ii) and (iii) ensure that the proposed dual reduction is valid with respect to the remaining constraints of an COP and the objective function. Condition (i) is more involved: it ensures that all jobs which do not belong to $\mathcal{J}'$ can never be assigned in a way such that the dual reduction is invalid. Therefore, it is required that for each job $i \in \mathcal{J} \setminus \mathcal{J}'$, there either exists a precedence condition between a job belonging to $\mathcal{J}'$ and job i or has a start time that is larger or equal to the largest earliest completion time of the jobs in $\mathcal{J}'$. Basically, the globally available information ensures that the posted constraint is valid.

In resource-constrained project scheduling problems with generalized precedence constraints (including RCPSP instances) only the artificial start time variable, which represents the makespan, is not monotonically non-decreasing w.r.t. the objective function (the objective coefficient is one). All other variables which are at most down-locked by the cumulative constraints are candidates for being part of $\mathcal{J}'$. In the special case of resource-constrained project scheduling problems these candidates also satisfy Condition (i) of Theorem 3.30. See Chapter 4 for a formal definition of this problem class.

Remark 3.31. A special case of Theorem 3.30 is the situation which is covered by Lemma 3.20. This is, the set of jobs $\mathcal{J}'$ contains a single job j with $\mathrm{ect} = \mathrm{ect}_j \leq \mathrm{hmin}$.

3.4.2 Set of cumulative constraints

The previous section considered the case of a single cumulative constraint. All results are generalizable for a set of cumulative constraints. We denote with $\text{hmin}(\mathcal{C})$ the first potential violated time point for a given cumulative constraint $\mathcal{C}$. The following corollaries map the results for a single cumulative constraint to multiple cumulative constraints.

Corollary 3.32 (Generalization of Lemma 3.20). Given a set $\mathfrak{C}$ of cumulative constraints that have a start time variable S_j in their scope, fixing a start time variable S_j to its earliest start time, that is $[\![S_j \leq \text{est}_j]\!]$, is dual feasible if the following conditions are satisfied:

(i) $\text{ect}_j \leq \text{hmin}(\mathcal{C})$ for all $\mathcal{C} \in \mathfrak{C}$,

(ii) only cumulative constraints $\mathcal{C} \in \mathfrak{C}$ down-lock S_j, and

(iii) the objective function f is monotonically non-decreasing in S_j.

Corollary 3.33 (Generalization of Lemma 3.24). Given a set $\mathfrak{C}$ of cumulative constraints that have a start time variable S_j in their scope, let B be the smallest time point where at least one of the cumulative constraints is potentially violated, i.e., $B = \min_{\mathcal{C} \in \mathfrak{C}} \text{hmin}(\mathcal{C})$. Adding the constraint

$$[\![S_j \leq \text{est}_j]\!] \cup [\![S_j \geq B + 1]\!]$$

is dual feasible if the following conditions are satisfied:

(i) only cumulative constraints $\mathcal{C} \in \mathfrak{C}$ down-lock S_j, and

(ii) the objective function f is monotonically non-decreasing in S_j.

Corollary 3.34 (Generalization of Theorem 3.26). Given a set $\mathfrak{C}$ of cumulative constraints that have a start time variable S_j in their scope, let us denote by $\mathcal{J}(\mathcal{C})$ the set of jobs which need to be scheduled w.r.t. constraint $\mathcal{C}$ and define $\text{ect} = \min_{\mathcal{C} \in \mathfrak{C}} \min_{i \in \mathcal{J}(\mathcal{C}) \setminus \{j\}} \text{ect}_i$. Adding the constraint

$$[\![S_j \leq \text{est}_j]\!] \cup [\![S_j \geq \text{ect}]\!]$$

is dual feasible if the following conditions are satisfied:

(i) only cumulative constraints $\mathcal{C} \in \mathfrak{C}$ down-lock S_j, and

(ii) the objective function f is monotonically non-decreasing in S_j.

Corollary 3.35 (Generalization of Theorem 3.30). Given a set $\mathfrak{C}$ of cumulative constraints and a precedence graph $D^{\text{P}} = (\mathcal{J}, A^{\text{P}})$ where $\mathcal{J}$ denotes the set of jobs which need to be scheduled, let $\mathcal{J}' \subseteq \mathcal{J}$ and $\text{ect} = \max_{\mathcal{C} \in \mathfrak{C}} \max_{i \in \mathcal{J}'} \text{ect}_i$. Adding the constraint

$$\bigcup_{j \in \mathcal{J}'} [\![S_j \leq \text{est}_j]\!]$$

is a dual feasible if the following conditions are satisfied:

(i) $\forall j \in \mathcal{J} \setminus \mathcal{J}'$ holds $\text{ect} \leq \max\{\text{hmin}, \text{est}_j\}$ or $\exists i \in \mathcal{J}'$ with $(i, j) \in A^{\text{P}}$,

(ii) only cumulative constraints $\mathcal{C} \in \mathfrak{C}$ down-lock the start time variables S_j with $j \in \mathcal{J}'$, and

(iii) the objective function f is monotonically non-decreasing for all start time variables belonging to the jobs in $\mathcal{J}'$.

Note that the used earliest completion time ect in the previous corollary depends on the cumulative constraints since these constraints define the processing time which is needed to compute the earliest completion time of job w.r.t. to a cumulative constraint.

3.5 Optional jobs

In this section we present presolving steps for the cumulative constraint with optional jobs as defined in Section 1.4.4. Most of the presolving steps introduced in this section are inspired by the resource allocation and scheduled problem which will be considered in Chapter 5 in which jobs have to be assigned to machines and all jobs are assigned to the same machine need to be scheduled with respect to a cumulative constraint. In Section 2.3 we presented linear relaxations for this type of constraint, shrinking the time windows of each job and removing irrelevant jobs from the scope of the constraint leads to potentially tighter linear relaxation (see Inequations (2.18), (2.19), and (2.23)).

All presolving steps introduced for the cumulative constraint without optional jobs can be applied to the case with optional jobs. To do this we assume that all potentially scheduled jobs are assigned to a resource. Due to the monotonicity of the inference performed, any redundant jobs detected under the all-jobs assumption remain redundant when a subset of jobs is assigned to a resource. That means, we can detect:

- ▷ situations where a cumulative constraint with optional jobs can be decomposed (Lemma 3.7);
- ▷ irrelevant jobs which can be removed from the scope of the constraint (Lemma 3.9 and Lemma 3.10);
- ▷ start time variables for which the down-lock or up-lock can be omitted (Lemma 3.11 and Corollary 3.12);
- ▷ start time variables which can be dual fixed to its earliest start time (Lemma 3.20) or latest start time (Corollary 3.21);
- ▷ start time variables for which certain domain values can be omitted (Lemma 3.24 and Corollary 3.25);
- ▷ schedule-or-postpone situations (Theorem 3.26 and Corollary 3.27).

In case the start time variables are only in the scope of a single cumulative constraint with optional jobs and do not contribute to the objective function, we can define a condition which implies that the constraint is redundant and leads to dual fixings. We assume (as before) that all potentially scheduled jobs are assigned to the resource. If the resultant single cumulative constraint has a feasible solution, the corresponding cumulative constraint with optional jobs is redundant because, independently of the assigned subset of jobs to the resource, a feasible schedule exists and the start time variables are only constrained by this single cumulative constraint with optional jobs. Any feasible solution of the resultant cumulative constraints is a feasible partial assignment for the whole underlying

Table 3.1: Overview of the dual reductions w.r.t. the effective horizon for a start time variable S_j which is in the scope of a (single) cumulative constraint.

Condition	Dual reduction	Reference
$\text{lct}_j \leq \text{hmin} \vee \text{est}_j \geq \text{hmax}$	remove variable from scope	3.9
$\text{lst}_j \leq \text{hmin} \wedge \text{ect}_j \geq \text{hmax}$	remove variable from scope	3.10
$\text{lst}_j \leq \text{hmin}$	omit down-lock for S_j	3.11
$\text{ect}_j \leq \text{hmin}$	$[\![S_j \leq \text{est}_j]\!]$	3.20
$\text{est}_j < \text{hmin}$	$[\![S_j \leq \text{est}_j]\!] \cup [\![S_j \geq \text{hmin} + 1]\!]$	3.24
$\text{ect}_j \geq \text{hmax}$	omit up-lock for S_j	3.12
$\text{lst}_j \geq \text{hmax}$	$[\![S_j \geq \text{lst}_j]\!]$	3.21
$\text{lct}_j > \text{hmax}$	$[\![S_j \geq \text{lst}_j]\!] \cup [\![S_j \leq \text{hmax} - p_j - 1]\!]$	3.25

problem. That allows the dual fixing of the start time variables to the solution values of a feasible solution which was computed under the all-jobs assumption. The variable locks (see Definition 1.12) allow us to identify such cumulative constraint with optional jobs where the start time variables are only in the scope of this constraint. Note that the binary choice variables can appear in other constraints and are allowed to contribute the objective function. The constraint integer programming formulations (Model 5.5) used for the resource allocation and scheduled problem which is considered in Chapter 5 contains such cumulative constraints with optional jobs.

3.6 Summary

We started the chapter by presenting the construction of a precedence graph using the global source of variable bounds (Algorithm 1). The precedence graph captures the information that certain jobs can only start if others are finished. This projection of the variable bound graph using the cumulative structure is used within several presolving techniques subsequently presented. Secondly, we analyzed the contribution of the cumulative constraint to the pools of global information, such as the variable locks and variable bounds. In the main part we developed two types of dual reductions: one which analyzes the setup w.r.t. the effective horizon, a concept we introduced in this dissertation, and the other which infers schedule-or-postpone situations. The effective horizon is also used to decompose a cumulative constraint (Lemma 3.7).

Table 3.1 gives an overview of the dual reductions w.r.t. the effective horizon. These reductions deal with jobs which are processed near the boundary of the effective horizon. Overall, if a start time variable S_j is only down-locked (up-locked) by one cumulative constraint, the objective function is monotonically non-decreasing (non-increasing) in S_j, and the feasible time window of job j is overlapping with hmin (hmax), at least one of these results is applicable. In Table 3.1 column "Condition" states the additional assumptions for the start time variable S_j. Column "Dual reduction" summarizes the implied dual reduction and the last column refers to the corresponding theorem, lemma, or corollary. Note that the conditions w.r.t. the variable locks and objective function are not required for the first two cases (Lemma 3.9 and Lemma 3.10) and the first case of hmin (Lemma 3.11) and hmax (Corollary 3.12), respectively.

Table 3.2: Overview of dual reductions related to schedule-or-postpone for a start time variable S_j which is in the scope of a (single) cumulative constraint, let $\text{ect} = \min_{i \in \mathcal{J} \setminus \{j\}} \text{ect}_i$ and $\text{lst} = \max_{i \in \mathcal{J} \setminus \{j\}} \text{lst}_i$.

Condition	Dual reduction	Reference
$\text{est}_j < \text{ect}$	$[\![S_j \le \text{est}_j]\!] \cup [\![S_j \ge \text{ect}]\!]$	3.26
$\text{lct}_j > \text{lst}$	$[\![S_j \ge \text{lst}_j]\!] \cup [\![S_j \le \text{lst} - p_j]\!]$	3.27

The effectiveness of these results can be classified w.r.t. the remaining problem as follows. Lemma 3.9 and 3.10 yield the strongest results since in both cases a variable is removed from the scope of a constraint. As this variable is not locked by that particular constraint anymore, dual information is provided to the remaining constraints. Then follows Lemma 3.11 and Corollary 3.12 which state a condition such that one of the two locks (down or up) can be omitted. Finally, the weakest result is given by the remaining lemmas and corollaries which require several additional conditions to be applicable. Note that these results are generalized w.r.t. a set of cumulative constraints (see Corollary 3.32 and 3.33).

Table 3.2 summarizes the schedule-or-postpone results for a single cumulative constraint. The columns are labeled in the same way as in Table 3.1 and the same additional conditions need to hold. Theorem 3.26 and Corollary 3.27 define requirements to discover schedule-or-postpone situations for a single start time variable within an arbitrary COP. Theorem 3.30 formalizes assumptions for a set of start time variable and proves that at least one of them can be fixed to its earliest start time.

4 Solving resource-constrained project scheduling problems

In the previous chapter we developed presolving techniques for cumulative constraints. These techniques are applicable to any COP that contains cumulative constraints. In this chapter we present a computational study which evaluates the impact of these presolving reductions for *resource-constrained project scheduling problems* [BDM+99]. The goal of resource-constrained project scheduling problems is to schedule jobs on (renewable) resources subject to precedence constraints such that the resource capacities are never exceeded and the latest completion time of all jobs is minimized. The cumulative constraint can be used to model renewable resources.

We use the constraint integer programming solver SCIP [Ach07b, Ach09] for our computational study. The following techniques are utilized to solve instances of resource-constrained project scheduling problems:

▷ during presolving, we use techniques discussed in Chapter 3 for cumulative constraints to reformulate the problem, more precisely the decomposition of cumulative constraints (Section 3.2), collecting structural information (Section 3.3), and dual reduction techniques (Section 3.4);

▷ during tree search, we apply the domain propagation algorithms time-tabling, edge-finding, and edge-finding time-tabling (see Section 1.4.3);

▷ infeasible sub-problems are analyzed to retrieve a conflict relaxation, therefore, the explanations introduced in Section 2.2.3 are used for inferences of cumulative constraints;

▷ the conflict relaxation is used to make branching decisions (see Section 2.2.2).

Contribution. This chapter contributes an extensive computational study of resource-constrained project scheduling problems using the constraint integer programming solver SCIP. Thereby, we focus on three topics:

▷ We evaluate the importance of the different propagation algorithms available for cumulative constraints for resource-constrained project scheduling problems.

▷ We analyze the impact of a presolving phase.

▷ We present results which compare SCIP against a state-of-the-art solver for resource-constrained project scheduling problems.

Previously published. The computational study for the impact of a presolving phase is joint work with Jens Schulz and J. Christopher Beck. Parts of these results were previously published in the following paper:

▷ Stefan Heinz, Jens Schulz, and J. Christopher Beck, *Using dual presolving reductions to reformulate cumulative constraints*, Constraints 18, no. 2 (2013), pp. 166–201.

Outline. This chapter is organized as follows. First we formally define the class of resource-constrained project scheduling problems in Section 4.1. In Section 4.2 we discuss the experimental setup which includes the computational environment, the considered test sets, and implementation details. The importance of the different propagation algorithms for resource-constrained project scheduling problems is evaluated in Section 4.3. The dual reductions for the cumulative constraint, which we discussed in Chapter 3, are considered in Section 4.4. Section 4.5 compares SCIP to a state-of-the-art solver for resource-constrained project scheduling problems. Finally, we conclude in Section 4.6 with a discussion.

4.1 Problem definition

Resource-constrained project scheduling problems with generalized precedence constraints are defined by a finite set $\mathcal{J} = \{1, \ldots, n\}$ of non-preemptive jobs and a finite set $\mathcal{R} = \{1, \ldots, m\}$ of renewable resources. Each resource $k \in \mathcal{R}$ has a bounded capacity $C_k \in \mathbb{N}$. Every job j has a processing time $p_j \in \mathbb{N}$ and resource demands $r_{jk} \in \mathbb{N}$ for each resource $k \in \mathcal{R}$. Note that the resource demand for certain job-resource combinations can be zero. The start time of a job is constrained by its predecessors, given by a precedence graph $D = (\mathcal{J}, A)$ and the distance function $\Delta : A \to \mathbb{Z}$. An arc $(i, j) \in A$ represents a "start-to-start" precedence relationship between two jobs. If $\Delta_{ij} > 0$, job j cannot start less than Δ_{ij} time units after job i started. If $\Delta_{ij} \leq 0$, job j can start at most $|\Delta_{ij}|$ time units before job i starts. In case $\Delta_{ij} = p_i$, we have the common precedence condition stating that job i must be finished before job j starts. The goal is to schedule all jobs with respect to resource capacities and precedence constraints, such that the makespan, i.e., the latest completion time of all jobs, is minimized. Instances which contain only common precedence conditions are called resource-constrained project scheduling problems (RCPSPs). In the case that generalized precedence conditions are present, we call these instances of the RCPSP/max problem.

A standard constraint programming model is the following:

$$\begin{array}{rll}
\min & \max\limits_{j \in \mathcal{J}} (S_j + p_j) & \\
\text{subject to} & S_i + \Delta_{ij} \leq S_j & \forall\, (i,j) \in A \\
& \texttt{cumulative}(\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}_{.k}, C_k) & \forall\, k \in \mathcal{R} \\
& S_j \in \mathbb{N} & \forall\, j \in \mathcal{J}.
\end{array}$$

The decision variable S_j for each job j defines the start time of the corresponding job and is bounded from below by zero. The objective function minimizes the maximum completion time of all jobs, i.e., the makespan. The linear constraints ensure that all precedence relationships are satisfied. Finally, the cumulative constraints enforce the capacity restrictions of the resources. Since the objective function is not linear, the above model is not a CIP. To retrieve a CIP we introduce an artificial variable which we minimize and bound from below by the completion time of all jobs. This results in the following model:

$$\begin{array}{rll}
\min & S_{n+1} & \\
\text{subject to} & S_j + p_j \leq S_{n+1} & \forall\, j \in \mathcal{J} \\
& S_i + \Delta_{ij} \leq S_j & \forall\, (i,j) \in A \\
& \texttt{cumulative}(\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}_{.k}, C_k) & \forall\, k \in \mathcal{R} \\
& S_j \in \mathbb{N} & \forall\, j \in \mathcal{J}.
\end{array}$$

There exist different MIP modeling approaches for the cumulative structure. For example, a time-indexed formulation which goes back to Pritskers et al. [PWW69, QS94], a flow-based formulation [AMR03], and a recently introduced event-based formulation [KALM11]. A computational study of these three models is presented in [KALM11]. None of these models, however, is competitive to current state-of-the-art approaches for this type of scheduling problems. One state-of-the-art approach is to combine propagation with conflict-driven search [SFSW11]. In particular, linear relaxations are not used. This is also the approach realized within SCIP. Hence, SCIP is used as a CP solver by disabling the linear programming relaxation.

Before we present our computational study, we discuss some aspects of the CIP model w.r.t. the global structures of variable bounds and variable locks. These are the two global structures which we combine with the cumulative structure within our presolving techniques. The linear constraints that describe the precedence conditions have a variable bound structure since the predecessor job bounds the start time of the successor job from below. As a result, the globally available variable bound graph contains the complete precedence graph of the resource-constrained project scheduling problem, allowing access to this information at any point during solving. The objective function of the CIP model contains only the artificial variable S_{n+1}. All other decision variables have an objective coefficient of zero. This implies that the objective function is monotonically increasing and decreasing in S_j for all $j \in \mathcal{J}$. The linear constraints which bound the artificial variable from below need to up-lock the start time variable S_j, whereas the artificial variable receives several down-locks (see Definition 1.12). For the linear constraints that describe the precedence relationships, each of the two start time variables is locked in one direction: for the predecessor, an up-lock is added and the successor gets a down-lock. Thus, start time variables belonging to jobs which do not have any predecessor do not receive any down-locks from the linear constraints. Only the cumulative constraints might need to down-lock these variables (see Section 3.3.1). Hence, the dual reductions for the cumulative constraint present in Section 3.4 are applicable.

4.2 Experimental setup

For our experiments, we use the constraint integer programming solver SCIP [Ach07b, Ach09]. In this section we introduce the chosen test sets and give some more details on SCIP and its application to resource-constrained project scheduling.

4.2.1 Test sets

For our experiments, we consider two test sets: RCPSP and RCPSP/max. In the following we introduce the selected instances for both classes and note the structural differences between them.

RCPSP test set

We use resource-constrained project scheduling problems with standard precedence constraints, for short RCPSPs. That is, for all $(i,j) \in A$, $\Delta_{ij} = p_i$. If $(i,j) \in A$ and $(j,i) \in A$, then the instance is trivially infeasible. Therefore, usually only one of these two arcs is present. The RCPSP is a suitable problem class to apply the dual reductions from Section 3.4 since there exist start time variables that have no down-locks (or have no up-locks) except, possibly, from the cumulative constraints.

We use the problem instances of the Psplib [KS96, PSP]. This library contains four categories, differing by the number of jobs to be scheduled: 30, 60, 90, or 120 jobs. The first three categories contain 480 instances each, the final one has 600 instances for a total of 2040 instances. Each category is clustered in classes of 10 instances and so there are 48 classes for the first three categories and 60 classes for the last category. Each instance contains four cumulative constraints.

Additionally, we use the pack instances [BP00, ADN08, HS11] which have the same type of precedence conditions. This test set contains 55 instances which are highly cumulative, meaning that the available resource capacity is much larger than the resource demands of the jobs. Hence, many jobs can be executed in parallel. This structure is not favorable for our dual reductions since the effective horizon is in most cases given by the smallest earliest start time and the largest latest completion time of all jobs.

These two sets together contain 2095 instances.

RCPSP/max test set

We also consider resource-constrained project scheduling problems with generalized precedence constraints (RCPSP/max). Informally, such a constraint represents a minimum or a maximum time that must elapse between the start of two jobs. In case both types exist for a pair of start time variables, down- and up-locks have to be placed on both variables. Therefore, we expect RCPSP/max to be less suitable for our dual reductions.

We select problem instances from the Psplib [KS96, PSP]. There are 12 test sets available. All instances have five cumulative constraints and differ in the number of jobs. The test sets are testsetc and testsetd each with 540 instances having 100 jobs; ubo10, ubo20, ubo50, ubo100, ubo200, ubo500, and ubo1000 each with 90 instances and 10, 20, 50, 100, 200, 500, and 1000 jobs, respectively; and j10max, j20max, and j30max each with 270 instances and 10, 20, and 30 jobs. In total there are 2520 instances.

4.2.2 Cumulative constraint handler

Within SCIP, constraints are implemented by constraint handlers in a similar fashion to Simpl [YAH10]. The cumulative constraint handler provides a variety of propagation algorithms and linear relaxations (see [BHL+10, BHS11, HS11]). We use the time-tabling propagation algorithm [LL82, KS99], perform an over-load checking via edge-finding [Nui94, Vil09a], run the edge-finding propagator, and the time-table edge-finding propagator [Vil11, SFS13]. Section 1.4.3 briefly describes these propagation algorithms and their realization within SCIP. In Section 4.3 we analyze the usefulness of these propagation algorithms for the resource-constrained project scheduling problems.

For our purposes, we extended the existing cumulative constraint handler. We implemented the presolving reductions discussed in Chapter 3 except the domain reduction stated in Theorem 3.30. In SCIP, variable domains are realized via single intervals and so domain holes cannot be represented. Therefore, the dual reductions presented in Lemma 3.24, Theorem 3.26, Corollary 3.33, and Corollary 3.34 are implemented via a probing step. This means, we discover such situations and tentatively try both branches. If one branch leads to an infeasibility we apply the other branch as a domain reduction.

4.2.3 Primal heuristics

In addition to the default SCIP components including the extended cumulative constraint handler, which are used with their default settings, we added a primal heuristic based on a fast list scheduling algorithm [LW92, MSSU03, KH06]. This primal heuristic is suitable for the RCPSP instances but not for the RCPSP/max instances and is executed during the presolving phase to find "good" feasible solutions.[1]

4.2.4 Parameters settings

SCIP is designed to solve problems by using a linear programming (LP) relaxation and so the default parameters are tuned under the assumption that (many) LP relaxations will be solved during search. In our case we do not solve a linear programming relaxation and, therefore, run the solver as a pure CP solver resulting in a much better performance for cumulative scheduling instances. Consequently, we used the predefined CP parameter settings[2] of SCIP as base settings for solving resource-constrained project scheduling problems. That implies that all features mentioned in Section 4.2.2 are enabled.

For MIPs, the LP relaxation is used (among other techniques) to guide the search. The binary and integer variables which have a fractional value in the LP solution (usually) form the candidate set from which a variable is selected for branching. Furthermore, the fractional values of these variables define the branching point for the selected variable. As a result of omitting the LP relaxation, this information are not available. The CP setting of SCIP, therefore, relies on a SAT-style conflict-driven search [MMZ+01]: conflicts which result from the analysis of infeasible sub-problems drive the search. We refer to Section 2.2.2 for more details.

The availability of a primal solution has an impact on the performance of the solver. In the case of resource-constrained project scheduling problems, any primal solution bounds the makespan variable from above and the precedence conditions propagate this bound to all other start time variables, leading to smaller domain sizes for the decision variables. Such pruning might also introduce cores for the corresponding jobs, see Section 1.4.3. Concerning the dual reductions we discussed in the previous chapter, smaller variable domains tend to be beneficial. To partly control performance variability arising through primal heuristics, we run most of our experiments with three different base settings corresponding to our expectations of worst, reasonable, and best-case situations for our techniques. Our base settings, summarized in Table 4.1, are as follows:

- ▷ NO-PRIMAL: All primal heuristics including the list scheduling heuristic are disabled. As a result, during the presolving phase, no primal solution is available and so the start time variables are unbounded for the RCPSP/max test sets and only bounded by the sum of all processing times for the RCPSP test sets. Such wide domains reduce the opportunities to make inferences.

- ▷ DEFAULT: SCIP runs with its default primal heuristics plus our problem specific scheduling heuristic.

[1] This primal heuristic is available within the "Scheduler" example of SCIP.

[2] In the interactive shell of SCIP, the CP solver settings can be set (and viewed) using the command `set emphasis cpsolver`. The method `SCIPsetEmphasis(scip, SCIP_PARAMEMPHASIS_CPSOLVER, TRUE)` does the same in the SCIP callable library.

Table 4.1: List and description of the used settings. As basis we are using the CP emphasis setting provided by SCIP. In addition we apply one of the base settings listed in this table.

setting	description
NO-PRIMAL	all primal heuristics are disabled
DEFAULT	enable the scheduling heuristic
BOUNDED	in addition to DEFAULT the value of the best known feasible solution is provided as objective limit

▷ BOUNDED: In addition to the DEFAULT settings, the start time domains are bounded by the objective value of the best known solution[3] for each instance. This condition substantially reduces the start time domains and increases the potential to find domain reductions during the presolving phase. The solver will only find solutions which have an objective value better than the provided best known value. If the given solution value matches the optimal solution value, the problem is infeasible.

4.2.5 Computational environment

All experiments were performed on Intel Xeon Core 3.20 GHz computers (in 64 bit mode) with 12 MB cache, running Linux, and 48 GB of main memory. We used SCIP version 3.0.1.4. For each instance we enforced a time limit of 1800 seconds.

4.3 Propagation algorithms

In Section 1.4.3 we discussed the propagation algorithms which are available in SCIP 3.0.1.4. In the following, we analyze the impact of these algorithms w.r.t. resource-constrained project scheduling problems. We examine the usefulness of each propagator for the RCPSP and for RCPSP/max.

In addition to the base settings described above, we enable and disable different propagation algorithms. First, we enable all propagation algorithms: time-tabling, edge-finding, and time-tabling edge-finding. Each algorithm performs a consistency check and tries to tighten variable domains. Second, we only enable the cheapest (w.r.t. worst case complexity) propagation algorithm. This is time-tabling. In addition to these two cases we run two more experiments. We enable one of the other two propagation algorithms edge-finding and time-tabling edge-finding in addition to the time-tabling algorithm. If a propagator is enabled it performs the consistency check and tries to shrink the variable domains. Note that all features mentioned in Section 4.2.2 are enabled.

Tables 4.3 and 4.5 present a summary of the overall results for RCPSP and RCPSP/max, respectively. For each base setting (NO-PRIMAL, DEFAULT, BOUNDED) and test set we state the number of solved instances (column "opt"), the number of instances for which a feasible solution was found (column "feas"), and the shifted geometric mean[4] for the number of search "nodes" and running "time" (in seconds) with shift $s = 100$ and $s = 10$, respectively. This ensures that outliers do not have a huge impact on the measures. Shifting similarly reduces the bias of easy instances, those solved in less than $s = 10$ seconds or requiring

[3]For the RCPSP instances we used the ones given in the PSPLIB [KS96, PSP] whereas for the RCPSP/max instances we considered the primal solutions in [SFSW13].

[4]The shifted geometric mean of values $t_1, \ldots, t_n$ is $\left(\prod(t_i + s)\right)^{1/n} - s$, with shift s.

Table 4.2: Number of instances which are easy and hard for the individual test sets of RCPSP. An instance is easy if all settings (see Table 4.3) solve an instance in less than one second. Hard instances are those which are not solved by any setting.

test set	J30	J60	J90	J120	PACK	total
evaluate	43	72	88	167	51	421
easy	437	359	315	125	1	1237
hard	0	49	77	308	3	437
total	480	480	480	600	55	2095

fewer than $s = 100$ search nodes. The first column “propagation” shows which propagators are enabled.

We removed those instances from the evaluation which are extremely easy or extremely hard to solve. Therefore, we follow the idea presented in [AW13]: We exclude instances for being too easy if they are solved by all settings in less than one second. Instances are declared as hard if none of the settings is able to solve them within the given time limit. Tables 4.2 and Table 4.4 state for the individual test sets belonging to RCPSP and RCPSP/max, respectively, the number of easy and hard instances. Any instance which does not belong to the hard instances was solved by at least one setting. Therefore, these tables give an overview on the number of instances which are solvable within the time limit.

4.3.1 RCPSP

In case of the RCPSP instances we have a basis of 2095 instances. From these 1237 are easy and 437 instances are hard leaving 421 instances for the evaluation. Table 4.2 presents the number of easy and hard instances for the individual test sets of the RCPSPs.

Given the number of instances which are part of the evaluation, Table 4.3 states the results for different propagation settings.

Test set j30. Independently of the chosen setting, for the J30 test set, all instances can be solved. Thereby, 437 instances out of 480 belong to the easy class, leaving 43 instances for the evaluation. As expected, the time-tabling algorithm alone needs the most search nodes. Running at least one of the expensive propagation algorithms in addition decreases the search nodes by a factor of more than two. Regarding the running time, the best result w.r.t. the overall running time for each base setting is observed with the use of the time-tabling algorithm together with time-tabling edge-finding. Providing the optimal solution value does not decrease the overall running time drastically, independently of the propagation setting.

Test sets j60, j90, and j120. For the three test sets J60, J90, and J120, the results are similar. The best setting w.r.t. the overall performance, independent of the base setting (NO-PRIMAL, DEFAULT, and BOUNDED), is to only use time-tabling and to disable both edge-finding and time-tabling edge-finding. The only exception is for the BOUNDED setting for J120 where the time-tabling and time-tabling edge-finding combination is almost 1 second (in shifted geometric mean) faster than the time-tabling setting. Applying any of the two propagation algorithms (edge-finding and time-tabling edge-finding) in addition to time-tabling often leads to a decrease in the number of search nodes. This, however, does not pay off w.r.t. the overall running time (except for J120 in case of the BOUNDED base setting). The number of solved instances varies only slightly within each base setting.

Table 4.3: Impact of the individual propagation algorithms which are available in SCIP w.r.t. RCPSP. For each base setting we state the number of instances solved (“opt”), the number of instances for which a feasible solution was found (“feas”), and the shifted geometric mean for the number of “nodes” and running “time” in seconds. Easy and hard instances are omitted from the evaluation (see Table 4.2).

		NO-PRIMAL				DEFAULT				BOUNDED			
propagation	test set	opt	feas	nodes	time	opt	feas	nodes	time	opt	feas	nodes	time
◦ time-tabling	J30	43	43	11125.4	7.7	43	43	10606.0	7.5	43	43	6264.7	6.4
	J60	71	72	21086.6	21.8	71	72	20155.0	21.7	72	72	5562.4	14.5
	J90	82	88	7399.7	17.1	81	88	5946.7	16.4	87	88	677.3	10.4
	J120	152	167	10940.6	17.4	150	167	9514.1	17.0	166	167	527.7	6.5
	PACK	14	51	3814565.0	1063.9	13	51	4113772.9	1092.0	14	51	3880417.1	1062.7
	RCPSP	362	421	23049.6	31.8	358	421	20778.2	31.4	382	421	3387.8	20.6
◦ time-tabling ◦ edge-finding	J30	43	43	4433.9	9.1	43	43	4249.4	9.1	43	43	2043.8	7.9
	J60	68	72	16232.5	40.8	69	72	16897.0	42.5	71	72	4259.1	25.7
	J90	78	88	7620.2	29.7	77	88	5732.6	28.1	82	88	600.1	14.8
	J120	150	164	9647.4	30.4	149	167	8977.9	31.3	163	167	292.5	6.6
	PACK	49	51	2897.7	7.7	50	51	2544.6	7.2	50	51	210.9	4.6
	RCPSP	388	418	8029.6	25.1	388	421	7260.1	25.2	409	421	673.1	10.4
◦ time-tabling ◦ time-tabling edge-finding	J30	43	43	4192.5	6.7	43	43	4292.5	6.8	43	43	2291.7	6.0
	J60	71	72	16700.8	27.9	71	72	16437.6	29.2	71	72	4325.8	18.9
	J90	79	88	8003.9	22.8	79	88	6157.3	22.3	85	88	613.3	12.4
	J120	151	166	9832.6	22.8	152	167	9466.9	23.3	164	167	286.8	5.6
	PACK	49	51	2882.6	6.4	50	51	2455.7	6.8	50	51	202.1	4.1
	RCPSP	393	420	8162.7	18.9	395	421	7466.7	19.2	413	421	679.7	8.5
◦ time-tabling ◦ edge-finding ◦ time-tabling edge-finding	J30	43	43	4433.9	9.2	43	43	4249.4	9.1	43	43	2043.8	7.9
	J60	67	72	16227.8	40.8	69	72	16893.4	42.5	71	72	4259.1	25.7
	J90	78	88	7628.8	29.6	77	88	5727.7	28.0	82	88	600.1	14.8
	J120	150	164	10557.7	31.2	149	167	8976.4	31.3	163	167	292.6	6.6
	PACK	49	51	2898.6	7.7	50	51	2545.0	7.2	50	51	211.0	4.6
	RCPSP	387	418	8324.6	25.4	388	421	7258.2	25.1	409	421	673.2	10.4

Having the best known objective value available for bounding the start time variable leads to the best performance, as expected. In case of enabling the time-tabling algorithm alone 431, 402, and 291 instances are solved for J60, J90, and J120, respectively (including the easy instances which are solved by all settings).

Test set pack. The picture changes for the PACK instances. For this test set it is crucial to use at least one of the expensive propagation algorithms edge-finding and time-tabling edge-finding in order to achieve small running times and a large number of solved instances. With at least one of them enabled in addition to time-tabling, we can solve 49 and 50 instances compared to 13 and 14 if only the time-tabling algorithm is used (ignoring the one easy instance). In the case of the time-tabling setting, the additional knowledge of a “good” or even optimal solution does not help. The overall performance is basically constant. This changes if edge-finding or time-tabling edge-finding are added to the set of propagators. Knowing the optimal solution value reduces the required search nodes by a factor of 10 and the overall running time by approximately 50%.

Summary. For all of the test sets, the three different base setting, NO-PRIMAL, DEFAULT, and BOUNDED, lead to the same conclusion. This indicates that the issue of random noise reported in [HSB13] for the same solver appears to have been reduced.

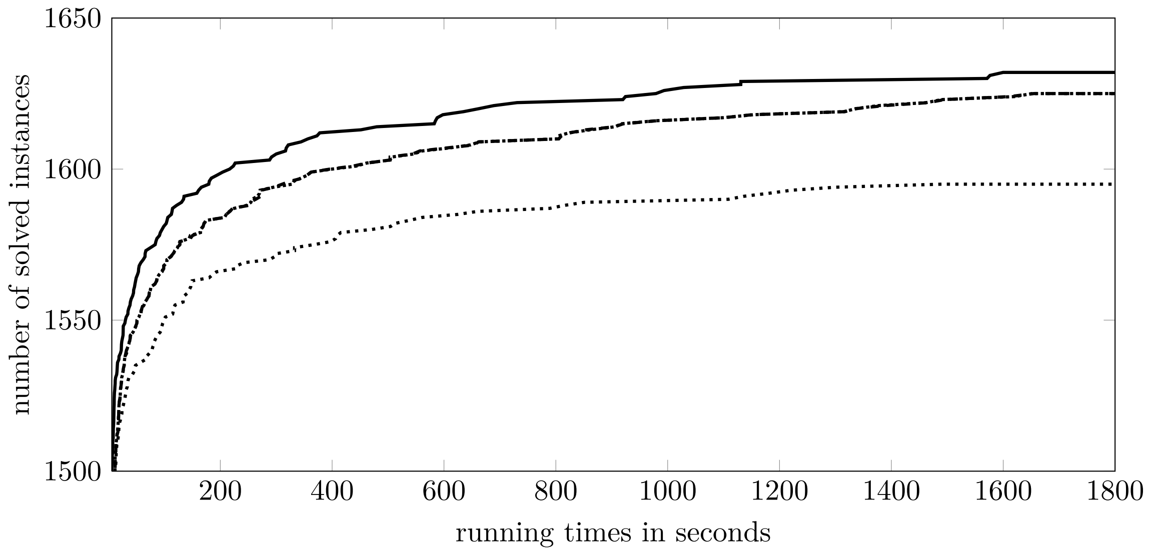


Figure 4.1: Performance diagram for the 2095 RCPSP instances w.r.t. the different propagation settings (the DEFAULT base setting is used). Showing the number of instances solved within a certain time. The time-tabling setting is dotted (······), the time-tabling and edge-finding setting is dashed (- - -), the time-tabling and time-tabling edge-finding setting is solid (——), and the time-tabling, edge-finding, and time-tabling edge-finding settings is densely dotted (········). Note that the performance results for time-tabling and edge-finding and time-tabling, edge-finding, and time-tabling edge-finding are very similar.

Figure 4.1 presents a performance diagram for the DEFAULT base setting. It visualizes the number of solved instances within a certain time for the four different propagator settings (see Table 4.3). Note that the two settings which enable the edge-finding propagator perform very similarly. The setting enabling the time-tabling and time-tabling edge-finding propagation algorithm leads to best performance w.r.t. this measure. It solves the largest number of instances within a fixed running time window. In contrast, the time-tabling propagator by itself solves the fewest number of instances. This difference, however, results mainly from the performance of these two settings for the PACK instances. For these instances the time-tabling and time-tabling edge-finding setting dominates the time-tabling setting by far (see Table 4.3).

For the test sets J60, J90, and J120, the results suggest, independently of the base setting, that the expensive propagation algorithms edge-finding and time-tabling edge-finding should be discarded. Enabling either of these algorithms often leads (as expected) to a decrease in the number of search nodes but an increase in the overall running time. The additional time spend in each search node for performing these algorithms does not pay off. For J30 the running time decreases slightly if time-tabling edge-finding is performed in addition to time-tabling. The number of solved instances is constant within the three base settings.

The insignificance of edge-finding and time-tabling edge-finding, stands in marked constrast with results presented for other solvers. In [Vil11, SFS13] results are discussed which indicate that time-tabling edge-finding is valuable for these problems. Even more, it is argued that this propagation algorithm is responsible for solving instances which had not been solved before. At the moment we are not able to give a proper reason for this disagreement. It could be due to a different implementation of the propagation algorithms

Table 4.4: Number of instances which are easy and hard for the individual test set of RCPSP/max. An instances is easy if all settings (see Table 4.5) solve an instances in less than one second. Hard instances are those which are not solved by any settings.

test set	j10max	j20max	j30max	TESTSETC	TESTSETD	UBO10	UBO20	UBO50	UBO100	UBO200	UBO500	UBO1000	total
evaluate	0	11	29	150	176	0	0	15	39	72	71	57	620
easy	270	259	241	366	354	90	90	73	40	0	0	0	1783
hard	0	0	0	24	10	0	0	2	11	18	19	33	117
total	270	270	270	540	540	90	90	90	90	90	90	90	2520

and the corresponding explanations or the tree search used. In Section 4.5 we compare our solver to the one presented in [SFS13].

For the PACK instances it seems to be crucial to use at least one of the expensive algorithms edge-finding and time-tabling edge-finding as this lead to much better performance compared to the time-tabling setting.

Overall, the results suggest using only the time-tabling propagation algorithm for RCPSP instances which are not highly cumulative. For instances which are highly cumulative (PACK test set) it seems to be valuable to use one of the more expensive (w.r.t. the worst case complexity) propagation algorithms, edge-finding or time-tabling edge-finding. Since disadvantages are small if time-tabling edge-finding is enabled with the time-tabling algorithm for instances which are not highly cumulative and the advantages for the highly cumulative instances are huge, the results suggest enabling time-tabling and time-tabling edge-finding for these instances as the default setting. This leads for the whole test set to the best performing w.r.t. the overall running time and the number of solved instances across all three base settings as indicate in Figure 4.1.

4.3.2 RCPSP/max

The RCPSP/max test set contains 2520 instances. Of these, 1783 instances are easily solvable by all settings in less than one second. Only 117 instances cannot be solved by any parameter setting and are classified as hard. This leaves 620 instances for the evaluation. Table 4.4 states the number of easy and hard instances for the different test sets. For the test sets J10max, UBO10, and UBO20 all instances fall into the easy category. Therefore, these sets are completely excluded from the analysis. The summarized results are given in Table 4.5. Before we analyze the results for different test sets, we want to point out, that the results for the NO-PRIMAL and DEFAULT base settings are very similar due to the fact that none of the primal heuristics, even the one we added to the heuristic pool, finds a primal solution for any of the instances. Differences which arise result in most cases from the primal heuristics, which are performed unsuccessfully, influencing statistical measures which lead once in a while to a (slightly) different search.

Test sets j20max and j30max. Most of the instances belonging to J20max and J30max are part of the easy class: 259 for J20max and 241 for J30max. For both test sets none of the instances are hard. Hence, only 11 and 29 instances for j20max and J30max, respectively, remain for the evaluation. The results suggest to use time-tabling together with time-tabling edge-finding. Doing this gives the best overall performance for all three base

Table 4.5: Impact of the individual propagation algorithms which are available in SCIP w.r.t. RCPSP/max. For each base setting we state the number of instances solved ("opt"), the number of instances for which a feasible solution was found ("feas"), and the shifted geometric mean for the number of "nodes" and running "time" in seconds. Easy and hard instances are omitted from the evaluation (see Table 4.4). In particular test sets J10max, UBO10, and UBO20 are completely omitted since all instances of these test sets are easy.

		NO-PRIMAL				DEFAULT				BOUNDED			
propagation	test set	opt	feas	nodes	time	opt	feas	nodes	time	opt	feas	nodes	time
○ time-tabling	J20max	11	11	40089.2	18.4	11	11	40089.2	18.5	11	11	30373.5	17.5
	J30max	20	28	95813.7	82.5	20	28	95692.8	82.7	20	28	71159.9	79.8
	TESTSETC	150	139	8425.8	16.3	150	139	8401.2	16.3	150	139	2345.4	11.5
	TESTSETD	171	168	4555.7	11.4	171	168	4557.5	11.4	175	168	1233.3	8.3
	UBO50	13	13	10726.7	26.4	13	13	10722.9	26.5	13	13	4726.2	23.8
	UBO100	37	34	1556.7	8.3	37	34	1556.4	8.3	38	34	337.4	5.9
	UBO200	72	62	527.5	3.5	72	62	527.5	3.5	72	62	44.1	1.1
	UBO500	70	60	1202.1	24.7	70	60	1202.1	24.8	71	60	216.4	8.3
	UBO1000	49	47	1190.0	222.1	49	47	1190.1	221.6	57	47	25.0	52.6
	RCPSP/max	593	562	3963.4	21.2	593	562	3960.6	21.2	607	562	1015.6	12.9
○ time-tabling ○ edge-finding	J20max	11	11	318.3	0.7	11	11	318.3	0.7	11	11	148.6	0.7
	J30max	29	28	3970.6	8.0	29	28	3970.6	7.9	29	28	1603.9	5.7
	TESTSETC	137	139	6920.2	39.2	137	139	6898.2	39.2	139	139	2026.8	26.6
	TESTSETD	163	168	3191.0	22.8	164	168	3191.6	22.8	165	168	850.8	15.4
	UBO50	15	13	3820.7	25.7	15	13	3820.7	25.6	14	13	1747.8	21.5
	UBO100	38	34	986.2	9.9	38	34	986.4	9.8	38	34	227.7	6.7
	UBO200	71	62	507.0	8.9	71	62	507.0	8.8	72	62	40.4	2.4
	UBO500	68	60	1277.3	98.2	68	60	1277.1	98.1	70	60	208.9	22.6
	UBO1000	36	45	1039.4	586.7	36	45	1039.0	584.8	48	47	25.4	63.8
	RCPSP/max	568	560	2437.5	38.5	569	560	2435.1	38.5	586	562	599.7	19.3
○ time-tabling ○ time-tabling edge-finding	J20max	11	11	307.3	0.6	11	11	307.3	0.6	11	11	140.7	0.5
	J30max	29	28	3585.1	5.0	29	28	3585.1	4.9	29	28	1346.3	3.1
	TESTSETC	144	139	7021.2	26.0	144	139	6999.5	26.1	146	139	2101.5	18.7
	TESTSETD	170	168	3626.9	16.0	170	168	3626.8	16.0	171	168	881.0	11.2
	UBO50	15	13	4061.3	15.7	15	13	4061.3	15.9	15	13	1642.3	14.3
	UBO100	38	34	1184.3	8.0	38	34	1184.2	8.0	38	34	232.1	5.1
	UBO200	72	62	524.3	5.8	72	62	524.3	5.8	72	62	40.3	1.7
	UBO500	69	60	1296.6	46.6	69	60	1296.5	46.5	70	60	208.0	14.2
	UBO1000	43	46	1092.5	320.3	43	46	1092.5	320.8	52	47	25.4	58.6
	RCPSP/max	591	561	2647.7	25.8	591	561	2645.7	25.8	604	562	617.7	14.3
○ time-tabling ○ edge-finding ○ time-tabling edge-finding	J20max	11	11	318.3	0.7	11	11	318.3	0.7	11	11	148.6	0.7
	J30max	29	28	3970.6	8.0	29	28	3970.6	8.0	29	28	1603.9	5.7
	TESTSETC	137	139	6920.9	39.2	137	139	6897.5	39.2	139	139	2026.6	26.5
	TESTSETD	164	168	3190.9	22.8	163	168	3191.5	22.8	165	168	850.6	15.4
	UBO50	15	13	3820.7	25.8	15	13	3820.7	25.8	14	13	1748.0	21.6
	UBO100	38	34	986.4	9.8	38	34	986.3	9.8	38	34	227.7	6.7
	UBO200	71	62	507.1	8.9	71	62	507.1	8.9	72	62	40.4	2.4
	UBO500	68	60	1277.2	98.2	68	60	1277.1	98.2	70	60	208.9	22.6
	UBO1000	35	45	1038.5	585.6	36	45	1039.4	586.7	48	47	25.4	63.8
	RCPSP/max	568	560	2437.5	38.5	568	560	2435.9	38.5	586	562	599.7	19.3

settings w.r.t. the running time and all instances are solved. Applying only the time-tabling algorithm leads to an increase in the number of search nodes compared to all other settings. For the test set J20max, 9 instances are no longer solved. Providing the optimal solution value to bound the start time variables from above results in the best performance (as expected) for all propagation settings. The number of search nodes is often halved.

Test sets testsetc and testsetd. For the test sets TESTSETC and TESTSETD, 150 and 179 instances out of 540 are neither easy nor hard. The best performance is achieved for all of the three base settings if only the time-tabling propagation algorithm is enabled. Adding either of the other two propagation algorithms leads to a decrease in the number of search nodes that is not matched by a decrease in the overall running time. In addition, the number of solved instances also decreases. All three base settings solve in total 516 instances of the TESTSETC, including easy instances, if the propagation algorithms are restricted to time-tabling. In case of TESTSETD, 529 instances are solved if the best bound is provided.

The ubo test sets. The UBO test sets have different number of jobs which need to be scheduled. They vary from 10 to 1000 jobs. For the test sets which have less than or equal to 100 jobs, it is beneficial to enable the propagation algorithms time-tabling and time-tabling edge-finding. For the test sets which contain more than 100 jobs, the additional time-tabling edge-finding propagator leads to an increase in the overall running time. In any case the edge-finding propagator does not improve the performance at all. If we bound the start time variables from above via the best known solution value, the best performance is achieved. The overall performance improves drastically in many cases.

Summary. The three base settings, NO-PRIMAL, DEFAULT, and BOUNDED lead to the same conclusion for the individual test sets. This indicates that the random noise which was reported in [HSB13] is reduced.

Figure 4.2 depicts a performance diagram for the DEFAULT base setting. It visualizes the number of solved instances, within a certain time, for the four different propagator settings (see Table 4.5). Note that the two settings which enable the edge-finding propagator perform very similarly. Using time-tabling leads to a slightly better performance (w.r.t. this measure) compared to the setting which enables the time-tabling and the time-tabling edge-finding propagation algorithm. It consistently solves a few more instances within a fixed time limit.

For the test set J20max and J30max, it is important to enable at least one of the expensive propagation algorithms edge-finding and time-tabling edge-finding besides the time-tabling algorithm. This leads to a huge decrease in the number of search nodes and an increase in the number of solved instances. For the other test sets, it seems to be reasonable to run the time-tabling algorithm alone as adding the time-tabling edge-finding algorithm leads to slightly worse results. For the whole test set, the results indicate that the time-tabling by itself is the best setting if the instances contain many jobs. This result holds w.r.t. the number of solved instances and overall running time. This setting is closely followed by a combination of time-tabling and time-tabling edge-finding.

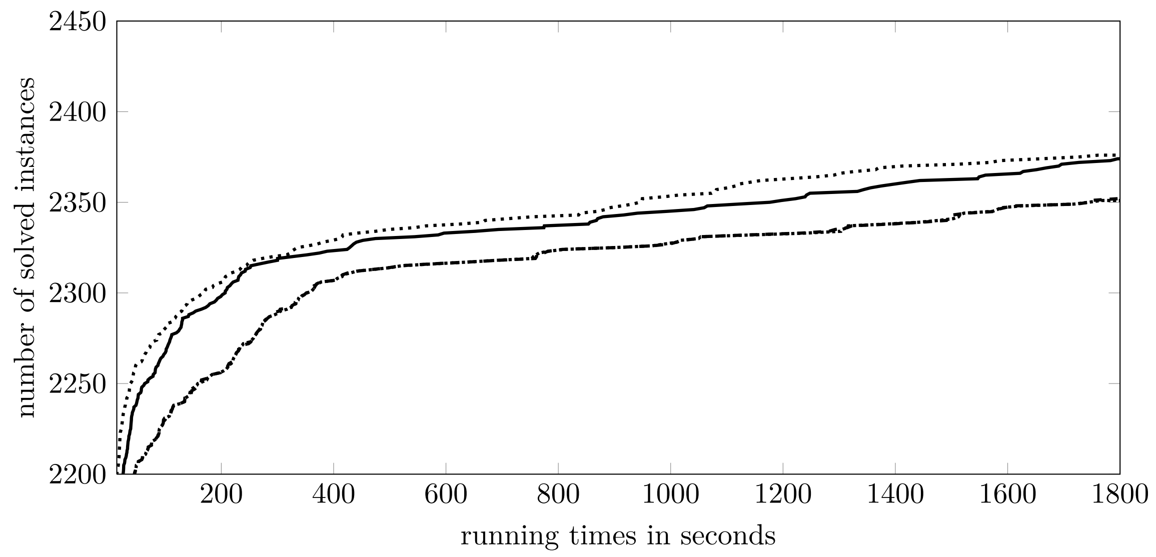


Figure 4.2: Performance diagram for the 2520 RCPSP/max instances set w.r.t. the different propagation settings. Showing the number of instances solved within a certain time. The time-tabling setting is dotted (⋯⋯), the time-tabling/edge-finding setting is dashed (- - -), the time-tabling/time-tabling edge-finding setting is solid (——), and the time-tabling/edge-finding/time-tabling edge-finding settings is densely dotted (⋯⋯). Note that the performance results for time-tabling/edge-finding and time-tabling/edge-finding/time-tabling edge-finding are very similar.

4.3.3 Overall summary

For both test sets, the three different base settings lead to the same conclusion w.r.t. the importance of the propagation algorithms which are available in SCIP. The results suggest omitting the edge-finding propagation algorithm completely independently of the test sets RCPSP and RCPSP/max. For those cases where the time-tabling algorithm alone is the most beneficial setting, adding time-tabling edge-finding leads to a slight increase in the running time and a few instances are no longer solved. Overall, however, it seems to be reasonable to enable time-tabling and time-tabling edge-finding. For the total RCPSP test set, this setting gives the largest number of solved instances and the best overall running time for all three base settings. For RCPSP/max, this combination is slightly inferior to the time-tabling alone.

Finally, we want to point out, that the insignificance of the edge-finding algorithm for these instances is surprising and contradictory to results achieved with other solvers running in different environments [Vil11, SFS13]. This should be further investigated.

4.4 Dual reductions

The study in [HSB13] presented an extensive computational study for the dual reduction techniques. The authors showed that for certain problem classes, a reduction of the problem, w.r.t. the number of variables can be achieved when dual reduction techniques are applied. However, there was no clear conclusion concerning the overall performance impact. The main issue was that the search suffered from too much random noise exacerbated by the strong dependence on the primal bound. We invested some effort to overcome this issue. In this section we rerun the experiments of [HSB13] utilizing a proper conflict-driven search (see Section 2.2.2).

To allow for comparison to the results presented in [HSB13] we perform exactly the same series of experiments. These experiments are set up to get an impression of the utilization of dual presolving steps, targeting the following questions:

1. How often are dual reductions made?
2. Does a primal solution increase the number and impact of the dual reductions?
3. Do the reductions increase the number of fixed variables after presolving?
4. How do these reductions contribute to the overall performance of the solver?

Therefore, we are using the base settings NO-PRIMAL, DEFAULT, and BOUNDED and run the solver with and without dual reductions for the cumulative constraint handler. Regarding the propagation algorithm, we enabled time-tabling and time-tabling edge-finding for consistency checks and variable domain reductions.

The results are presented in Tables 4.6–4.10. The first three tables state results w.r.t. the presolving phase. Tables 4.9 and 4.10 show the overall impact of the presolving steps on problem solving performance.

For RCPSP/max instances, typically no primal solution is found during the presolving phase, hence, the results for the NO-PRIMAL setting are similar to the DEFAULT setting. As already mentioned, differences arise due to side effects of unsuccessfully running primal heuristics. Primal heuristics influence statistical measures which are used for branching.

For the PACK instances (part of the RCPSP test set), none of the dual reductions were applicable during presolving. As expected, the highly cumulative structure of these instances results, in most cases, in an effective horizon that is equivalent to the interval defined by the smallest earliest start time and the largest latest completion time over all jobs. Hence, none of the dual reductions is applicable.

In the following, we present our results in detail.

4.4.1 Applicability of dual reductions

Table 4.6 presents information about the number of inferences made by each implemented presolving reduction in the base setting (NO-PRIMAL, DEFAULT, BOUNDED). The first column "test set" states the name of test set. The remaining columns display the following information for each setting, that is NO-PRIMAL, DEFAULT, and BOUNDED. The first column "inst" is the percentage of problem instances where at least one reduction was made. The percentage is taken w.r.t. all instances of the test set, including the PACK instances. There are 2095 and 2520 instances for the RCPSP and RCPSP/max test sets, respectively. The second column prints the "total" number of times the presolving reduction was applied. The remaining columns display, for those instances for which the reduction was applied at least once, the maximum ("max") number of times it was applied for a single instance, the average ("avg") number, and the standard deviation ("var"). If no reductions were found for the whole test set we print "–". Note that the total number of times a reduction is applied depends on the "size" of the reduction. The solver may solve an instance during the presolving phase via a small number of large domain reductions whereas in another case the solver may make more, smaller reductions but yet fail to solve the problem in the presolving phase. So while the number of reductions made is indicative of whether the conditions required for inference occur, in general a smaller total number for one setting compared to another does not mean that this setting performs less total inference.

Table 4.6: This table summarizes the appearance of the different presolving reductions.

	NO-PRIMAL					DEFAULT					BOUNDED				
test set	inst	total	max	avg	var	inst	total	max	avg	var	inst	total	max	avg	var
Constraint decompositions (Lemma 3.7)															
RCPSP	0.0%	–	–	–	–	14.0%	775	11	2.6	2.1	16.8%	1685	53	4.8	5.8
RCPSP/max	6.3%	1205	64	7.6	9.5	6.3%	1205	64	7.6	9.5	33.6%	11022	592	13.0	28.9
Irrelevant variables due to no overlap with the effective horizon (Lemma 3.9)															
RCPSP	67.0%	51363	324	36.6	64.9	58.9%	18579	194	15.1	19.2	37.5%	21286	364	27.1	45.5
RCPSP/max	26.3%	17484	616	26.4	49.5	26.3%	17484	616	26.4	49.5	44.1%	192732	10849	173.5	487.4
Irrelevant variables due to an overlap with the effective horizon (Lemma 3.10)															
RCPSP	0.0%	–	–	–	–	1.8%	56	4	1.5	0.8	4.7%	248	20	2.5	3.1
RCPSP/max	0.4%	19	4	2.1	1.1	0.4%	19	4	2.1	1.1	11.7%	887	44	3.0	4.6
Variable lock adjustments (Lemma 3.11)															
RCPSP	0.0%	–	–	–	–	17.7%	2420	64	6.5	7.2	20.3%	4241	118	10.0	13.5
RCPSP/max	3.9%	908	67	9.3	13.1	3.9%	908	67	9.3	13.1	33.5%	12474	431	14.8	28.8
Dual fixings due to a single constraint (Lemma 3.20, Lemma 3.24, and Theorem 3.26)															
RCPSP	48.6%	9955	80	9.8	14.5	34.8%	3839	58	5.3	5.6	21.0%	3212	59	7.3	7.9
RCPSP/max	10.4%	570	31	2.2	2.7	10.4%	570	31	2.2	2.7	17.0%	839	18	2.0	1.7
Dual fixings due to a set of constraints (Corollary 3.32, Corollary 3.33, and Corollary 3.34)															
RCPSP	64.0%	16975	86	12.7	20.3	48.4%	3869	24	3.8	3.4	28.0%	1633	19	2.8	2.4
RCPSP/max	21.4%	2845	45	5.3	7.2	21.4%	2845	45	5.3	7.2	22.8%	2924	46	5.1	5.3
All dual reductions															
RCPSP	85.1%	78293	409	43.9	78.2	66.9%	29538	227	21.1	25.9	40.7%	32305	444	37.9	58.9
RCPSP/max	29.1%	23031	673	31.4	58.9	29.1%	23031	673	31.4	58.9	49.9%	220878	11614	175.7	497.6

Before we start to discuss the individual dual reductions and their impact, we provide a first comparison to the results presented in [HSB13]. There we collected the same statistics with an earlier version of the solver. The results for the base setting NO-PRIMAL are very similar. Some of the dual reductions are now applied more often, especially for the RCPSP/max test sets. The same holds for the base settings DEFAULT and BOUNDED regarding the test set RCPSP/max. However, for the test set RCPSP it seems that the picture changes in case of DEFAULT and BOUNDED, most importantly for the irrelevant variable detection and dual fixings. Here the number of instances where these reductions are applicable is smaller than in our previous results [HSB13]. Analyzing this carefully revealed that now many instances are solved during the first presolving round when a "good" solution is given. Even better, the primal reductions made by the propagation algorithms are sufficient to prove optimality. As a result the dual reduction algorithms are not called at all. In the previous results these instances where also solved during the presolving phase. Often it took, however, several rounds of presolving to finally prove optimality. Hence, our dual reduction algorithms where successfully called. Taking general improvement of the used solver into account the results presented here still have the same flavor as before.

Decomposition

In case of the decomposition (see Lemma 3.7), the availability of a primal solution appears essential. A "good" primal solution bounds the start time variables from above, leading to further domain reductions by the propagation algorithms. Shrinking the feasible time window of jobs further increases the applicability of other dual reductions that move jobs out of the effective horizon, increasing the chance of finding points in time within the effective time horizon where the capacity is never exceeded. The overall applicability of Lemma 3.7 is, however, small compared to the total number of instances. For the RCPSP test set only 14% and 17% of the instances are affected in case of the DEFAULT and BOUNDED setting, respectively. For the RCPSP/max test set 6%, 6%, and 33% of the instances are reformulated if the NO-PRIMAL, DEFAULT, and BOUNDED setting is applied.

Compared to the results presented in [HSB13] this dual reduction was more often applied here, especially for the RCPSP/max test set. A reason for that is that we added a presolving step which searches for redundant disjunctive constraints (see Section 3.3.2).

Irrelevant jobs

Irrelevant jobs are those that run completely before or after the effective horizon (Lemma 3.9) or have to be processed during the entire effective horizon (Lemma 3.10). The first type of irrelevant jobs appears quite often for the RCPSP test set, independent of the experimental setting: on average up to 36 start time variables are irrelevant over the whole set of cumulative constraints. Note that a start time variable that is irrelevant for several cumulative constraints is counted once for each of these cumulative constraints.

For the RCPSP/max test set, our experimental results indicate that this reduction is applicable to fewer instances in case of the base setting NO-PRIMAL and DEFAULT. For the BOUNDED case, however, a higher percentage of instances are affected compared to the RCPSP instances. On instances where this dual reduction is applicable, there is substantial inference. On one instance in the BOUNDED condition, 10849 reductions were triggered.

Knowing a “good” or even an optimal solution decreases the number of identified irrelevant jobs for the RCPSP test set. This heavily differs from the results presented in [HSB13]. In [HSB13] the number of identified irrelevant jobs increased if a “good” or even an optimal solution was given. A careful analysis of this issue revealed the following. Many instances which are solved during the presolving phase are solved within the first presolving round before the detection of irrelevant jobs is conducted.

For the RCPSP/max test set, the availability of a “good” primal solution increases the success of detecting irrelevant jobs. This increase is again related to the more narrowly bounded start time variables. In those instances where the reduction is applicable, the average number of applications is low compared to the maximum. The variance is moderate compared to the maximum, indicating that most of the instances have a similar number of irrelevant jobs and only for a few instances the particular reduction can be applied heavily. Note that the high percentage of irrelevant variables per constraint is not induced by the decomposition since the percentage is already high in the setting NO-PRIMAL, where no decomposition takes place for the RCPSP instances and only a few for the RCPSP/max problems.

The second type of irrelevant jobs (Lemma 3.10) only occurs (in these test sets) if a cumulative constraint is decomposed. These irrelevant jobs can be seen as an inference arising from the decomposition. The results are similar to those presented in [HSB13].

Variable locks

The results for removing variable locks (Lemma 3.11) are similar to those for decomposition (Lemma 3.7). It is essential to have a “good” primal solution to trigger this propagation. Otherwise, the time window of a job is too large compared to its processing time, dramatically reducing the possibility of removing a variable lock because the condition that the latest start time of a job is smaller than hmin is unlikely to be satisfied. A closer look at the instances reveals that the applicability of this reduction is not necessarily related to those instances where a decomposition of a cumulative constraint takes place. In most cases, a decomposition leads to variable lock deletions. However, there are instances where locks are removed even though no decomposition was found.

Comparing these results to the one presented in [HSB13] shows an increase in the number of times this technique is applicable. This, however, does not follow from the newly introduced presolving step which adds redundant disjunctive constraints to increase propagation (see Section 3.3.2). Since these disjunctive constraints are redundant, they do not need to be checked for feasibility. Within SCIP, this implies that these constraints do not lock any of their variables. The increase in the number of times this dual reduction was applied is a result of the stronger primal reductions we are now doing compared to the results in [HSB13].

Dual fixings

The applicability of the dual fixing conditions (Lemma 3.20, Lemma 3.24, Theorem 3.26, Corollary 3.32, Corollary 3.33, and Corollary 3.34) seems to be (almost) independent of having a primal solution for the RCPSP/max test sets. In our previous results [HSB13] this also held for the RCPSP instances. Due to the heavy increase in the number of instances which are now solved trivially in the first presolving round if a "good" or even optimal solution is given, the percentage of instances where this technique is applied is smaller for the base setting DEFAULT and BOUNDED compared to previously published results.

These reductions, the ones related to a single cumulative constraint (Lemma 3.20, Lemma 3.24, and Theorem 3.26) and the ones related to multiple cumulative constraints (Corollary 3.32, Corollary 3.33, and Corollary 3.34), are applicable quite often for the RCPSP test set when no feasible solution is known (NO-PRIMAL). Note that within our implementation the single constraint dual fixing is done first, followed by the set-based version. As a consequence, the reductions stated for a set of cumulative constraints are those that the single constraint algorithm could not find because more than one cumulative constraint locked a particular variable. The total number of dual fixings from these two algorithms is therefore the sum of the corresponding numbers in the two rows. Similar to the irrelevant variable results, these reductions are less effective for the RCPSP/max test set.

The single constraint case is applicable for almost half of the RCPSP test set in case of the NO-PRIMAL base setting and fixes on average almost 10 variables. For RCPSP/max, reductions are found on more than 10% of instances with an average of 2 variable fixings independently of the base setting. Performing dual fixings on sets of constraints further increases the number of fixed variables during presolving by up to almost 13 variables on average for the RCPSP test set and 5 variables for the RCPSP/max test set. Hence, much smaller problems, in terms of the number of variables, remain after presolving. We evaluate this in more detail below.

It is notable that considering a set of cumulative constraints increases the applicability of the dual reductions. These fixings cannot be made by propagating a single constraint. In SCIP this type of propagation algorithm is natural since constraints of one type are all known to the corresponding constraint handler for that constraint type. Integrated reasoning about sets of constraints of the same type is similar to what is done in SIMPL [YAH10] but does not seem to have been otherwise investigated in the CP literature.

Summary

These results show that there are instances within the test sets of the PSPLIB that are easily solvable via dual reductions. For example, the maximum value for the applied dual fixings (Table 4.6) shows that for some instances almost all variables are fixed during presolving. Furthermore, knowing a primal solution is helpful since it bounds the start time variables from above leading to situations where instances are trivially solvable in the first presolving round without applying our dual reduction algorithms. This explains the decrease in the number of instances which are affected by our dual reductions for the RCPSP instances where a primal solution is known during the presolving phase (DEFAULT and BOUNDED) compared to previous results published in [HSB13].

Overall, the introduced presolving steps result in inferences for 85% of the instances for the base setting NO-PRIMAL and the RCPSP test set. For the DEFAULT and BOUNDED case this percentage is 66% and 40% for this particular test set. The number of irrelevant variables per constraint is 9 on average (recall that these instances have 4 cumulative constraints) and the number of dual fixings is more than 6 (combining the dual fixings due to a single and a set of constraints). For the test set PACK, none of the dual reductions was successfully applied. This is not surprising since these instances are highly cumulative: many jobs can be processed in parallel since their resource demands are small compared to the available resource capacity. This results in an effective horizon which matches the interval given by the smallest start time and the latest completion time of all jobs.

As expected, for the RCPSP/max test set, the reductions are not as effective. Still these reductions do arise in 28% of the instances if there is no primal solution and 47% of the instances if an objective limit (BOUNDED) is given. In both cases, on average up to 5 variables are dual fixed. Overall, the dual reductions are more often applied compared to the results presented in [HSB13] due to stronger primal reductions.

4.4.2 Presolving impact

The introduced presolving steps are applicable quite often for the RCPSP test set and arise frequently for the RCPSP/max test set. Applicability, however, does not mean that these reductions provide any new information or solving power. Therefore, Tables 4.7 and 4.8 present, for each test set of RCPSP and RCPSP/max, respectively, information about the number of *additional* variables fixed after presolving due to our proposed dual presolving steps, compared to when there are no cumulative constraint dual reductions. Recall that SCIP has a number of default presolving techniques [Ach07b] and so these experiments test whether our new techniques actually result in more inference than what SCIP is already capable of in presolving. The columns in the tables have the same meaning as in Table 4.6. The issue that instances are solved trivially in the first presolving round does not affect these because we measure the presolving impact after presolving is completed. Thereby, it does not matter in which presolving round an instance is solved. The results shown here are similar to the previously stated [HSB13].

RCPSP

If no primal solution is available (NO-PRIMAL), the new dual reduction propagation algorithms are able to shrink the problem size for a great portion of the instances. Fixings occur in 367, 429, and 456 instances of the test set with 30, 60, and 90 jobs, respectively, containing 480 instances each. For the test set with 120 jobs the problem size is additionally reduced for 530 instances out of 600. For the first three test sets, the average number

Table 4.7: Effect for the 2095 RCPSP instances of the introduced presolving steps w.r.t. the number of *additional* variables that are fixed during the presolving phase. We omit the PACK test set because no additional variables were fixed.

	NO-PRIMAL					DEFAULT					BOUNDED				
test set	inst	total	max	avg	var	inst	total	max	avg	var	inst	total	max	avg	var
30 jobs	76.5%	4636	31	12.6	13.1	48.5%	1052	31	4.5	4.8	16.2%	378	31	4.8	7.2
60 jobs	89.4%	8935	61	20.8	25.3	60.2%	1570	31	5.4	4.8	17.1%	304	61	3.7	6.8
90 jobs	95.0%	13197	91	28.9	37.4	65.2%	2167	73	6.9	7.1	20.2%	385	91	4.0	9.5
120 jobs	88.3%	2801	32	5.3	4.6	87.7%	2750	35	5.2	4.6	52.3%	1353	102	4.3	8.3
RCPSP	85.1%	29569	91	16.6	25.3	65.0%	7539	73	5.5	5.4	27.3%	2420	102	4.2	8.2

of additionally fixed variables is around one third of the total number of variables. For the largest test set we have an average of 5.3 variables fixed. The maximum number of additionally fixed variables indicates that there is at least one instance within each of the sets J30, J60, and J90 for which all variables are fixed due to dual reductions.

However, if a primal solution is known, the number of instances with additional fixed variables after the presolving phase is smaller compared to the case where no primal solution is known. The reason for this can be seen in Table 4.9 which is discussed below in more detail. If no primal solution is available and the dual reductions are omitted, none of the 2095 instances is solved during the presolving phase. Existence of a primal solution increases the number of instances solved in presolving (even if the dual reductions are disabled). For all such instances, the number of unfixed variables after presolving is zero and, hence, it is not possible to fix additional variables. Taking this into account, we can conclude that when a "good" primal solution is known, the dual presolving steps are able to fix on average between 3 and 6 additional variables.

RCPSP/max

The impact of the dual reductions differs over the different test subsets. For TESTSETC and TESTSETD, we see a similar impact as for the RCPSP instances. With no primal solution (NO-PRIMAL), 254 and 216 instances of TESTSETC and TESTSETD (each set contains 540 instances), respectively, are additionally reduced. On average 7 and 6 additional variables are fixed. If a primal solution is available (BOUNDED case), we observe the same phenomenon as for the RCPSP instances: the number of instances with additional fixings decreases. For these two test sets, the reason is not due to instances solved during the presolving phase (see Table 4.10) but it is related. Due to the presence of a primal bound, the domains of the start time variables are reduced, triggering propagation algorithms which find many of the fixings. Hence, the dual reductions provide only a small number of additional fixings.

For the test sets J10, J20, and J30 (each having 270 instances), a negligible number of instances (37, 14, and 10, respectively) are affected when no primal solution is given. For these few instances only one variable is additionally fixed. If a primal bound is given, the number of affected instances decreases again and the average number of fixed variables increases by a small amount for J10 and J30 and decreases by a small amount for the other test set J20.

For the UBO test sets (each of 90 instances) the impact is also negligible. Only between 0 and 12 instances are influenced by dual reductions, independent of the setting. For these few instances, only 1 to 5 variables are additionally fixed in average. There is one outlier in test set UBO1000 where the dual reductions lead to solve one additional instance.

Table 4.8: Effect for the 2520 RCPSP/max instances of the introduced presolving steps w.r.t. the number of *additional* variables that are fixed during the presolving phase.

	NO-PRIMAL					DEFAULT					BOUNDED				
test set	inst	total	max	avg	var	inst	total	max	avg	var	inst	total	max	avg	var
J10max	13.7%	61	5	1.6	1.0	13.7%	61	5	1.6	1.0	4.1%	28	11	2.5	3.0
J20max	5.2%	19	3	1.4	0.6	5.2%	19	3	1.4	0.6	5.6%	20	3	1.3	0.6
J30max	3.7%	15	3	1.5	0.8	3.7%	15	3	1.5	0.8	1.1%	7	3	2.3	0.9
TESTSETC	47.0%	1791	59	7.1	8.6	47.0%	1791	59	7.1	8.6	5.7%	156	61	5.0	11.1
TESTSETD	40.0%	1337	60	6.2	8.5	40.0%	1337	60	6.2	8.5	7.2%	116	17	3.0	3.5
UBO10	13.3%	17	2	1.4	0.5	13.3%	17	2	1.4	0.5	1.1%	1	1	1.0	0.0
UBO20	7.8%	11	2	1.6	0.5	7.8%	11	2	1.6	0.5	4.4%	12	7	3.0	2.3
UBO50	1.1%	1	1	1.0	0.0	1.1%	1	1	1.0	0.0	3.3%	7	5	2.3	1.9
UBO100	1.1%	1	1	1.0	0.0	1.1%	1	1	1.0	0.0	6.7%	31	21	5.2	7.2
UBO200	2.2%	2	1	1.0	0.0	2.2%	2	1	1.0	0.0	4.4%	10	6	2.5	2.1
UBO500	0.0%	–	–	–	–	0.0%	–	–	–	–	1.1%	2	2	2.0	0.0
UBO1000	0.0%	–	–	–	–	0.0%	–	–	–	–	1.1%	1001	1001	1001.0	0.0
RCPSP/max	22.0%	3255	60	5.9	8.1	22.0%	3255	60	5.9	8.1	4.7%	1391	1001	11.7	91.3

Summary

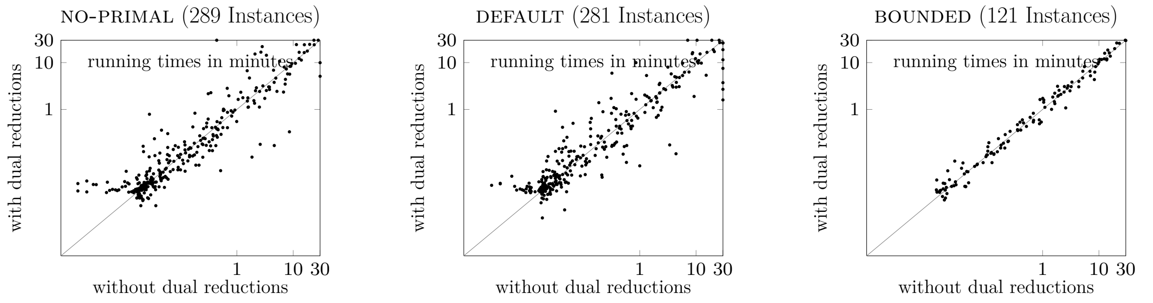
For both test sets, the tables show that the introduced presolving steps provide additional domain filtering: substantially more for the RCPSP test set than for RCPSP/max. Since the worst case complexity of the standard propagation algorithms and the subsequently required search depend on the number of jobs, these reductions indicate the potential for a speed-up in problem solving. If such a speed-up can be observed is analyzed in the next section.

4.4.3 Overall impact

In Tables 4.9 and 4.10, we present results that indicate the impact of the introduced presolving steps w.r.t. the overall performance. For these computations, we enforced a time limit of 1800 seconds for each instance. We applied the predefined CP settings (see Section 4.2.4 for more details) and enabled the propagation algorithms time-tabling and time-tabling edge-finding. For each test set and setting, we report several results for the case the dual reductions were omitted ("without dual reductions") and the dual reductions were applied ("with dual reductions"). First we state, in column "inst", the percentage of instances for which at least one of the dual reductions was applied. Second we print in column "solved" the number of instances that were solved and proved optimal within the time limit. The next two columns "pres" and "root" show the number of instances that were solved in the presolving phase and during root node processing of the search tree: the number of instances solved within the presolving phase is included in the root node number. Finally, we state the number of instances that are solved after 1 second, 1 minute, 5 minutes, and 10 minutes. We do not report any aggregated run-time measures since the number of instances that are solved quickly and the ones that are not solved at all dominate the results. Instead we show performance diagrams w.r.t. the running times (logarithmic scale). We only depict a result if at least one algorithm took longer than one second to solve the instance and at least one algorithm solved the instance within the given time limit. The corresponding number of instances is stated in the title of each figure. Hence, we omitted trivial and unsolvable instances.

Table 4.9: Impact of the dual reductions on the overall performance for the 2095 instances of RCPSP. The performance diagrams compare the running time for using the dual reductions (with dual reductions) and omitting them (without dual reductions) for the three base setting NO-PRIMAL, DEFAULT, and BOUNDED (see Table 4.1).

		without dual reductions							with dual reductions						
setting	inst	solved	pres	root	1 sec	1 min	5 min	10 min	solved	pres	root	1 sec	1 min	5 min	10 min
NO-PRIMAL	85.1%	1428	0	0	1231	1387	1408	1418	1427	360	360	1218	1387	1405	1415
DEFAULT	66.9%	1043	12	24	848	1005	1025	1034	1047	15	32	837	1002	1025	1038
BOUNDED	40.6%	509	221	221	428	473	490	497	510	231	231	427	472	491	496



RCPSP

Table 4.9 shows the results for the 2095 instance of the RCPSP test set. It is notable that in case no primal solution (NO-PRIMAL) is available, the dual reductions provide enough information that 360 instances are solved directly in the presolving phase. For the base setting DEFAULT and BOUNDED 3 and 10 instances are additionally solved during the presolving phase if dual reductions are allowed.

Overall, 4 and 1 additional instances are solved within the time limit in case of the DEFAULT setting and BOUNDED setting. For the NO-PRIMAL setting the dual setting fails to solve 1 instance that is solved when the dual reductions are omitted. The performance figures are not in favor of either of the two settings. From these results, it is not possible to conclude that either condition is dominating. They indicate that for the overall performance the dual reductions are neutral.

RCPSP/max

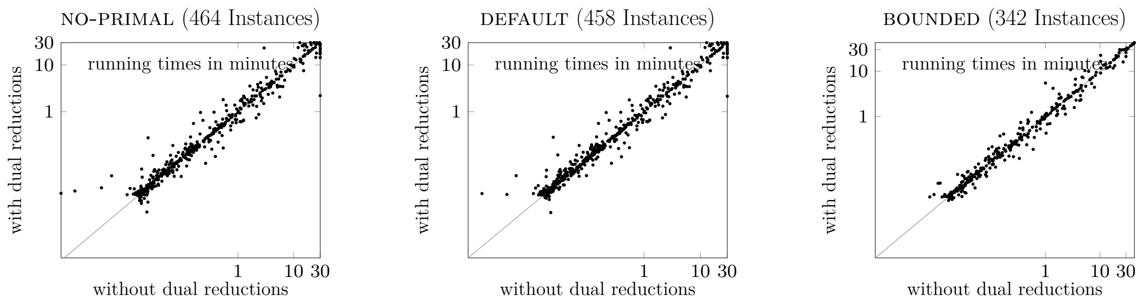
The results for the overall impact of the dual reductions for the RCPSP/max test set are given in Table 4.10. Here we get the same picture as for the RCPSP instances: the performance figures and numbers in the table do not indicate that one of the settings is favorable. The dual reductions have a neutral impact on the overall performance for RCPSP/max instances.

Summary

Overall, a few instances are additionally solved within the time limit using the dual reductions. This number of instances, however, is small compared to the size of the test sets. Overall, the impact of the dual reductions w.r.t. running time can be seen as neutral, independent of the setting. This result is consistent with the fact that only a few jobs (between 1 and 6) are additionally fixed using the DEFAULT setting. On the positive side, given that we have removed feasible (or even optimal) solutions from the solution space due to the

Table 4.10: Impact of the dual reductions on the overall performance for the 2520 instances of RCPSP/max. The performance diagrams compare the running time for using the dual reductions (with dual reductions) and omitting them (without dual reductions) for the three base setting NO-PRIMAL, DEFAULT, and BOUNDED (see Table 4.1).

		without dual reductions							with dual reductions						
setting	inst	solved	pres	root	1 sec	1 min	5 min	10 min	solved	pres	root	1 sec	1 min	5 min	10 min
NO-PRIMAL	28.7%	723	132	132	687	717	723	723	723	132	132	688	719	723	723
DEFAULT	28.7%	723	132	199	690	717	723	723	723	132	200	688	719	722	723
BOUNDED	47.2%	1162	740	740	1083	1148	1158	1160	1162	741	741	1079	1148	1157	1160



dual reductions, the overall results show that the solver is still able to solve most of the instances as before.

Without a primal solution during the presolving phase (NO-PRIMAL), the dual reductions allow for solving 360 instances in the presolving phase that could not be solved in the presolving phase before.

4.5 Comparision to state-of-the-art solvers

In the remainder of this chapter, we compare our implementation against a state-of-the-art solver for RCPSP and RCPSP/max. We choose a solver which improved the best known results for instances of the PSPLIB recently [SFS13, SFSW13]. The authors[5] of these papers provided two executables which they used for their recent experiments. Their solver is called lazy clause generator (LCG) and works in a similar fashion as SCIP for cumulative scheduling instances. As the name of the solver states, clauses are generated lazily during the search. Therefore, a SAT relaxation is created which is fed with all bound changes and their explanations. If an infeasibility is reached, a SAT-based conflict analysis is performed and conflict clauses are created and added to the relaxation. For a more detailed description of the solver we refer to [FS09].

Note that we received two different problem specific solvers. One for the RCPSP instances and one for the RCPSP/max instances. Having these executables allows us to compare LCG and SCIP within the same environment.

For the comparison we run both solvers in the same environment and applied a time limit of 1800 seconds. In case of SCIP we used the predefined CP settings (see Section 4.2.4 for more details) and enabled the propagation algorithms time-tabling and time-tabling edge-finding. On top of that we applied our dual reduction steps during the presolving phase.

[5]Thanks to Andreas Schutt and Peter Stuckey for providing the executables and helping the understand the solver output.

Table 4.11: Comparison of the performance of SCIP and LCG for the 2095 RCPSP instances. In addition to these two solvers we state the results for the virtual best solver (VBS). We present two sets of results which differ in the instances which are removed from the evaluation since they are declared to be hard.

(a) An instances is declared to be hard if both solvers hit the time limit and fail to solve this instance.

				SCIP				LCG				VBS			
test set	total	easy	hard	opt	feas	nodes	time	opt	feas	nodes	time	opt	feas	nodes	time
J30	480	438	0	42	42	2893.5	6.9	42	42	4311.6	1.5	42	42	2255.5	1.2
J60	480	344	43	86	93	8578.0	30.5	93	93	6494.1	8.6	93	93	5163.8	8.3
J90	480	240	78	154	162	878.4	8.7	161	162	1033.7	4.5	162	162	561.7	3.8
J120	600	107	314	170	179	5193.0	15.9	165	179	5711.6	12.7	179	179	2298.4	6.5
PACK	55	2	3	49	50	2437.3	7.0	16	50	1696254.2	571.8	50	50	2202.1	5.2
RCPSP	2095	1131	438	501	526	2960.1	13.6	477	526	5927.9	14.6	526	526	1743.8	5.3

(b) If at least one solver does not solve an instances, this instance is seen to be hard. In other words, only instances which are solved by both solvers are part of the evaluation.

				SCIP				LCG				VBS			
test set	total	easy	hard	opt	feas	nodes	time	opt	feas	nodes	time	opt	feas	nodes	time
J30	480	438	0	42	42	2893.5	6.9	42	42	4311.6	1.5	42	42	2255.5	1.2
J60	480	344	50	86	86	5402.7	19.7	86	86	4074.2	4.0	86	86	3197.0	3.8
J90	480	240	87	153	153	542.9	4.6	153	153	660.0	2.2	153	153	344.7	1.9
J120	600	107	337	156	156	3189.5	9.1	156	156	2369.6	3.1	156	156	1276.3	2.5
PACK	55	2	38	15	15	5765.8	11.0	15	15	342628.7	37.7	15	15	4885.4	6.6
RCPSP	2095	1131	512	452	452	2009.5	8.9	452	452	2176.4	3.4	452	452	1116.3	2.5

Table 4.11 and Table 4.12 present the summarized results for RCPSP and RCPSP/max, respectively. For each test sub-set we state first the number of "total" instances belonging to the test set, followed by the columns showing the number of instances which are "easy" and "hard". An instance is classified as easy if both solvers solve this instance in less than one second. For each test set we present two result tables which differ in the definition which instances are considered to be hard. First we declare an instance to be hard if both solvers fail to solve this instance. Second we additionally add an instance to the hard class if at least one solvers fails to solve this instance. Instance which do not belong to the easy and hard class are part of the evaluation and aggregated measures are reported.

For each solver (SCIP and LCG) we state the number of instances solved to proven optimality (column "opt"), the number of instances for which a feasible solution was found (column "feas"), and the shifted geometric mean for the search "nodes" and running "time" in seconds. Again we shift the time by 10 seconds and the search nodes by 100. In addition to the individual results for the solvers, we present the results for the so-called virtual best solver (VBS). This means that for each instance we choose the solver which performs best w.r.t. the running time. For the VBS we state the same measures as for the two solvers.

RCPSP

SCIP solves 1632 instances (1131 easy instances and 501 instances which belong to the instances under investigation) of 2095 instances, while LCG solves 1608 instances (including the 1131 easy instances). Both solvers together prove optimality for 1656 instances. The number of instances which are solved differs only slightly and the additional instances solved by the virtual best solver is small as well. This indicates that the two solvers have

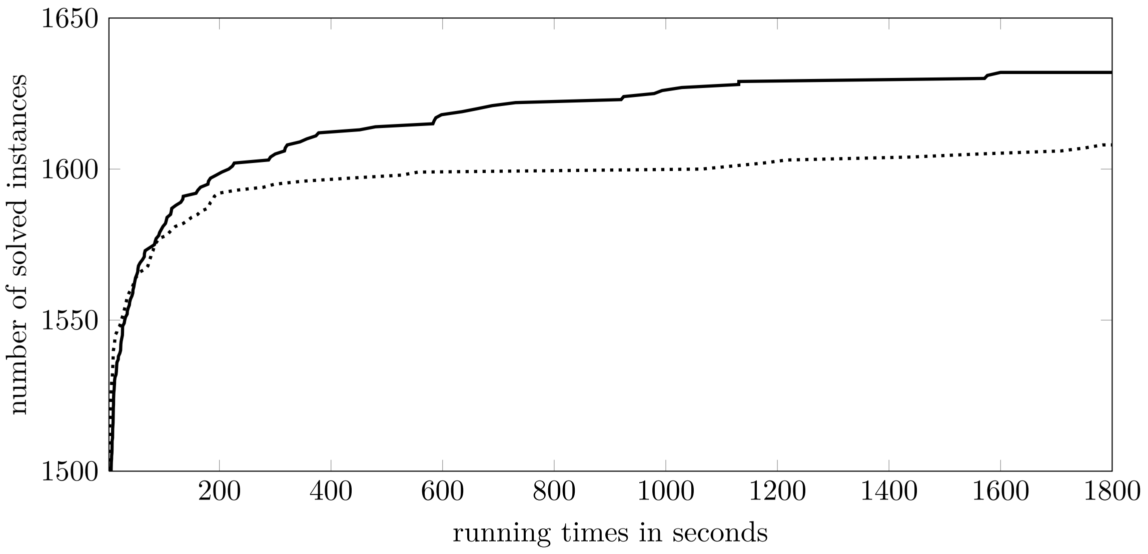


Figure 4.3: Performance diagram for the 2095 RCPSP test set comparing the two solvers SCIP and LCG. The result for SCIP is depict by a solid line (——). The LCG result is shown as a dotted line (······).

a similar performance. Regarding the overall performance, LCG is faster by a factor of more than 2. For the complete test set SCIP need 8.9 seconds compared to 3.4 seconds (in shifted geometric mean) in case we only consider the instances which are solved by both solvers (Table 4.11(b)). Both solvers are able to find a feasible solution for each instance.

For the individual test set, SCIP performs much better for the PACK instances, solving 51 of 55 instances. LCG is able to prove optimality for 18 instances. For the 15 instances which are solved by both solvers, SCIP needs almost a factor of 60 fewer nodes. This gives a factor of more than 3 for the overall running time by which SCIP is faster than LCG. Only one instances is solved by LCG where SCIP fails.

Both solvers are able to solve all instances which contain 30 jobs (test set J30). From 480 instances, belonging to this test set, 438 are easy and can be solved by both solvers in less than one second. For the remaining 42 instances SCIP need only two thirds of the search nodes (in shifted geometric mean) compared to LCG. The number of search nodes visited by SCIP is only slightly larger compared to the virtual best solver. This indicates that SCIP needs consistently fewer search nodes to solve these instances. W.r.t. the running time this does not pay off as LCG is a factor of 4 faster than SCIP.

For the test sets J60 and J90, LCG solves a few more instances within the given limit than SCIP: 7 instances for each test set. Again LCG is clearly faster than SCIP. The results of the virtual best solver show that the instances solved by SCIP are a proper sub-set of those solved by LCG.

For the instances where 120 jobs need to be scheduled, SCIP solves 277 instances where LCG solves 272 instances. SCIP is able to succeed on 5 more instances than LCG. The VBS results shows that both solvers fail on instances where the other succeeds. Again LCG is a factor of 3 faster on those instances which both solvers solve (Table 4.11(b)).

Obviously, SCIP tends to need more time per search node which could be a reason for trailing slightly behind LCG in the number of solved instances for the test sets J60 and J90. On the other hand, SCIP dominates on the PACK test set. The version of LCG we used features both propagation algorithms time-tabling and time-tabling edge-finding, the same

Table 4.12: Comparison of the performance of SCIP and LCG for the 2520 RCPSP instances. In addition to these two solvers we state the results for the virtual best solver (VBS). We present two sets of results which differ in the instances which are removed from the evaluation since they are declared to be hard.

(a) An instances is declared to be hard if both solvers hit the time limit and fail to solve this instance.

				SCIP				LCG				VBS			
test set	total	easy	hard	opt	feas	nodes	time	opt	feas	nodes	time	opt	feas	nodes	time
J10max	270	259	0	11	9	22.5	1.0	11	9	49.3	1.0	11	9	21.0	1.0
J20max	270	251	0	19	15	106.9	1.0	19	15	1316.1	1.8	19	15	103.9	1.0
J30max	270	226	0	44	36	1022.4	3.2	40	36	7948.2	21.2	44	36	882.3	2.1
TESTSETC	540	322	9	188	199	4238.2	26.8	209	199	3422.5	7.6	209	199	2468.3	7.3
TESTSETD	540	323	0	201	212	2748.7	17.7	217	212	2673.0	5.1	217	212	1793.3	4.9
UBO10	90	87	0	3	2	17.4	1.0	3	2	35.9	1.0	3	2	14.5	1.0
UBO20	90	89	0	1	1	48.0	1.0	1	1	26.0	1.0	1	1	26.0	1.0
UBO50	90	67	1	21	19	2835.5	19.4	22	19	5266.9	12.9	22	19	2471.1	12.2
UBO100	90	25	6	53	48	1116.3	13.2	59	48	2899.2	9.5	59	48	989.9	8.4
UBO200	90	0	17	72	63	567.7	6.8	73	63	1266.4	6.1	73	63	468.0	3.9
UBO500	90	0	19	69	61	1375.6	49.3	70	61	2028.1	30.9	71	61	841.2	24.4
UBO1000	90	0	29	43	45	1259.1	485.3	58	58	2702.4	78.9	61	58	722.1	78.4
RCPSP/max	2520	1649	81	725	710	1855.2	24.7	782	723	2635.3	11.1	790	723	1257.0	9.1

(b) If at least one solver does not solve an instances, this instance is seen to be hard. Meaning only instances which are solved by both solvers are part of the evaluation.

				SCIP				LCG				VBS			
test set	total	easy	hard	opt	feas	nodes	time	opt	feas	nodes	time	opt	feas	nodes	time
J10max	270	259	0	11	9	22.5	1.0	11	9	49.3	1.0	11	9	21.0	1.0
J20max	270	251	0	19	15	106.9	1.0	19	15	1316.1	1.8	19	15	103.9	1.0
J30max	270	226	4	40	32	1112.4	3.5	40	32	4005.3	10.8	40	32	947.0	2.3
TESTSETC	540	322	30	188	178	2297.7	13.8	188	178	1839.4	3.1	188	178	1330.0	2.8
TESTSETD	540	323	16	201	196	1759.9	9.9	201	196	1779.8	2.5	201	196	1162.9	2.3
UBO10	90	87	0	3	2	17.4	1.0	3	2	35.9	1.0	3	2	14.5	1.0
UBO20	90	89	0	1	1	48.0	1.0	1	1	26.0	1.0	1	1	26.0	1.0
UBO50	90	67	2	21	18	2091.3	14.2	21	18	3868.9	8.6	21	18	1807.3	8.0
UBO100	90	25	12	53	42	533.0	4.1	53	42	1386.3	2.8	53	42	480.6	2.0
UBO200	90	0	18	72	62	524.3	5.8	72	62	1151.0	5.2	72	62	429.8	3.1
UBO500	90	0	22	68	60	1335.9	43.7	68	60	1671.1	25.5	68	60	803.1	21.2
UBO1000	90	0	50	40	40	2541.0	243.3	40	40	1510.0	45.5	40	40	1224.6	45.5
RCPSP/max	2520	1649	154	717	655	1331.6	14.0	717	655	1670.3	6.1	717	655	896.2	4.9

propagation algorithms which are enabled in SCIP. Therefore, these results are surprising since the dual reductions we developed do not lead to any additional reduction on the PACK instances.

Finally, Figure 4.3 shows a performance diagram for both solvers. This diagram visualizes the number of solved instances within a fixed time limit. The figure suggest that SCIP dominates LCG w.r.t. this measure. This domination, however, results from the different performance on the PACK instances.

RCPSP/max

Overall SCIP solves 2374 of 2520 instances where LCG is able to solve 57 more instances. The results of the virtual best solver (VBS) show that SCIP solves 8 instances which are not solved by LCG. Regarding the search nodes, SCIP tends to need slightly fewer nodes than LCG. As for the RCPSP instances, this does not lead to a better overall performance.

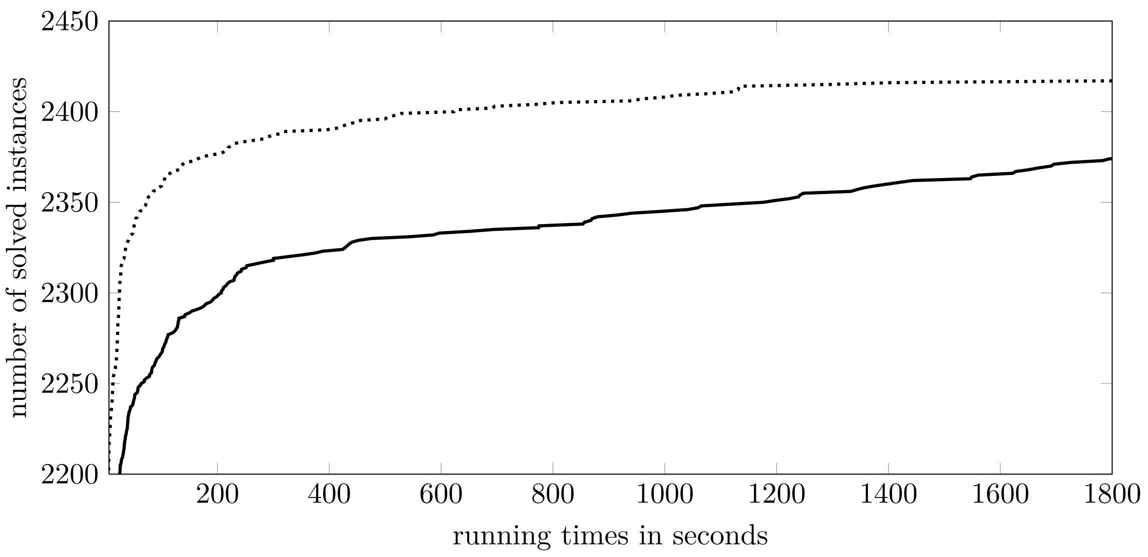


Figure 4.4: Performance diagram for the 2520 RCPSP/max instances comparing the two solvers SCIP and LCG. The result for SCIP is depict by a solid line (——). The LCG result is shown as a dotted line (······).

LCG is factor of more than 2 faster than SCIP. SCIP needs 14 seconds where LCG only requires 6.1 seconds (in shifted geometric mean) for those instances which are solved by both solvers (see Table 4.12(b)).

Most of the instances belonging to J10max, J20max, and J30max are easy and none of them are hard. For the evaluation, 11, 19, and 44 instances are left for J10max, J20max, and J30max, respectively. SCIP is able to solve all 810 instances. LCG hits the time limit on 4 instances belonging to J30max. For these three test sets SCIP does not only need fewer search nodes it also is faster in the overall performance.

Only 9 instances of the test set TESTSETC are found to be hard. The remaining 531 instances are solvable with LCG. For the test set TESTSETD all 540 instances are solved by LCG. SCIP proves optimality for 510 and 524 within the time limit of 1800 seconds for the instances belonging to TESTSETC and TESTSETD, respectively. For these instances LCG is more than a factor 3 faster than SCIP.

For the UBO test sets we have a similar picture as for the J60 and J90 test sets. SCIP trails slightly behind LCG w.r.t. the number of solved instances. For the test sets UBO10, UBO50, UBO100, UBO200, and UBO500, SCIP has a better overall performance than LCG.

Analyzing the results of SCIP reveals that several instances which have a large number of jobs hit the time limit during the presolving phase. This is due to the presolving steps which search for redundant disjunctive constraints (see Section 3.3.2) and redundant variable bounds (see Section 3.3.3). These consume too much time. This needs to be evaluated more carefully and should be adjusted in future releases.

Finally, Figure 4.4 shows a performance diagram for both solvers. This diagram visualizes the number of solved instances within a fixed time limit. This figure indicates that SCIP tends to be (slightly) slower w.r.t. the overall performance compared to LCG.

Summary

The comparison to state-of-the-art solvers for RCPSP and RCPSP/max indicate that our general purpose implementation is competitive to problem specific solvers. For the PACK instances which are part of the RCPSP test base, SCIP is even superior to LCG. For the remaining instances belonging to this class, LCG is faster w.r.t. the running time but can only solve a few more instances than SCIP. Taking the complete RCPSP test set into account SCIP is able to solve 24 more instances than LCG. A similar picture is indicated by the results for the RCPSP/max instances. Again there are test sets where SCIP dominates LCG and vice versa. For this type of problem LCG solves few more instances than SCIP. In any case SCIP can be considered as state-of-the-art solver for cumulative scheduling problems.

Overall, SCIP tends to use fewer search nodes compared to LCG which does not pay off w.r.t. the overall performance. The comparison shows that SCIP spends much more time within a search node than LCG. Since we do not have access to the implementation of LCG we only can provide some high-level technical explanations for this difference. These explanations should be taken with care since we are currently not able to analyze the corresponding impact on the overall running time. SCIP is designed to solve problems through the heavy use of an LP relaxation. In the case of cumulative scheduling we disabled this feature completely. However, the corresponding infrastructure needs to be maintained, requiring time during the traversal of the search tree. In addition SCIP is not limited in the way of the traversing the search tree. As a result, data structures are required to be able to “jump” through the tree, particularly for the conflict analysis. In contrast LCG performs conflict-directed clause learning (CDCL) [BHvMW09] allowing for much more efficient data structures for the search tree and conflict analysis. This could be a reason for LCG needing less time per node. Again these high-level arguments have to be taken with care as they are not based on empirical results.

4.6 Summary

In our first series of experiments, we evaluated the impact of the different propagation algorithms which are available in SCIP (see Section 1.4.3) w.r.t. their importance for resource-constrained project scheduling problems. The edge-finding propagation algorithm in its implementation in SCIP is not important at all. Disabling this algorithm leads to a better performance for both test sets (RCPSP and RCPSP/max). This is surprising since in the literature [Vil11, SFS13] it is reported that this algorithm is crucial to solve some of the instances of the PSPLIB. In a much milder way the same observation was made for the time-tabling edge-finding algorithm. Enabling this algorithm with the basic time-tabling propagator leads to a slight performance loss. For the complete RCPSP test set, however, the largest number of instances is solved with time-tabling and time-tabling edge-finding together across all three base settings: NO-PRIMAL, DEFAULT, and BOUNDED.

In a second set of experiments, we evaluated the impact of different dual reductions for resource-constrained project scheduling problems, utilizing variable locks and the effective horizon. These two related concepts can be exploited to create dual reduction techniques for cumulative constraints. Our first experiment shows that these reductions often occur for RCPSP instances and less frequently for the RCPSP/max problems. The results further show that our approach is able to find additional variable fixings during the presolving phase (Tables 4.7 and 4.8). In particular, for RCPSP instances with 30, 60, and 90 jobs about 33% of the variables are additionally fixed, in contrast to disabling the proposed dual reductions. For the RCPSP instances from PSPLIB, between 20% and 30% of the variables per cumulative constraint are irrelevant and can be removed from the scope of the constraint (Table 4.3). This provides a theoretical speed-up as the worst case complexity of the search depends exponentially on the number of jobs and the worst case complexity of standard propagation algorithms depends heavily on the number of job.

However, removing feasible solutions from the solution space can have negative consequences as it may be more difficult for the solver to find any feasible solutions at all. This effect is well-known in CP where it has been shown that symmetry breaking constraints can conflict with variable ordering heuristics [Kiz04]. Similarly, Borrett and Tsang [BT01] observe that reformulating a model can have a negative impact depending on the algorithms used. The results in Tables 4.9 and 4.10, however, indicate that there is no overall increase in solving time for the investigated instances: the dual reductions are neutral w.r.t. the overall running time.

Our results show that several instances of the RCPSP test set are easily solvable even without the presence of a primal solution which bounds the makespan variable from above. For these instances, our dual reduction techniques safely (w.r.t. completeness) fix start time variables to their lower bound. For highly cumulative instances, such as the PACK instances, these techniques do not apply since many jobs can be executed in parallel and exceed the capacity when running together: the effective time horizon cannot be tightened. Our dual reduction techniques, therefore, provide a better understanding of easy and hard instances. These techniques are able to remove, in some sense, the easy part from a cumulative constraint.

From a broader perspective and despite the significant work in CP on symmetry breaking, the use of presolving and dual reductions is not yet a standard component of constraint solvers. In contrast, for MIP solvers, presolving techniques are crucial for state-of-the-art performance. Our experimental results show that between 27% and 85% (depending on the settings) of the RCPSP instances could be additionally reformulated during presolving and sometimes to the point of solving the problem to optimality without search (see Table 4.7). For the RCPSP/max test set between 4% and 22% of the instances are further reformulated (see Table 4.8). We believe that these numbers alone indicate that presolving and dual techniques are exciting directions and opportunity for constraint solving research.

Finally, we compared the capability of SCIP, as a cumulative scheduling solver, against a problem specific state-of-the-art implementation. The results clearly showed that SCIP can be considered as a state-of-the-art solver for resource-constrained project scheduling problems in its standard version as well as with generalized precedence constraints.

5 Solving resource allocation and scheduling problems

In this chapter we consider scheduling problems with optional jobs. That means in a first phase the decision if a job will be processed on a particular resource needs to be made. Whereas, in a second phase all jobs assigned to the same resource have to be assigned start times. We restrict ourselves to a resource allocation and scheduling problem in a basic version: given a finite set of jobs, for each job a (renewable) resource has to be selected where the job gets processed. A job has different costs, resource demands, and processing times (which are non pre-emptable) for the different resources. The goal is to assign each job to a resource such that for each resource a feasible schedule exists and the assignment cost is minimized. A state-of-the-art approach to solve such problems is a logic-based Benders decomposition (LBBD) [Hoo04, Hoo05a, Hoo07, CCH13] that decomposes the problem into an assignment problem and a scheduling problem. In a first phase an assignment of jobs to resources is constructed which minimizes the assignment cost. In a second phase the assignment is checked for feasibility w.r.t. the scheduling constraints. If it is feasible, an optimal solution is found. Otherwise, a Benders cut is created which is added to the assignment problem to forbid the assignment. Then the algorithm returns to the first phase. It has been shown [Hoo05b, Hoo05a, Hoo07, CCH13] that this approach performs orders of magnitude better than a constraint programming approach or a mixed-integer programming model.

In this chapter we utilize the linear relaxation constructed for the cumulative constraint with optional jobs in Section 2.3 and present an improved LBBD approach and mixed-integer programming approach. As a third approach we develop a constraint integer programming model which takes advantage of the cumulative constraint with optional jobs. For each of these approaches we show computational results to identify the strengths and weakness of the different formulations. Finally, we compare the three approaches.

Contribution. In this chapter we present the following contributions for the resource allocation and scheduling problems. In Section 5.3 we use the linear relaxation for the cumulative constraint with optional jobs, developed in Section 2.3, for an improved sub-problem relaxation in a LBBD approach. Empirical results show that the improved sub-problem relaxation is the best choice. It is inexpensive to compute and does not lead to a slow down. In Section 5.4 we discuss mixed-integer programming formulations for the problem under investigation. We recall an model for this problem and analyze its behavior for modern MIP solvers. As a result we suggest an extended model which is superior to previously known models. For the considered test instances, the extended MIP formulation gives a speed-up factor of almost five compared to the basic MIP model. In Section 5.5 we present a constraint integer programming model which use the cumulative constraint with optional jobs.

Previously published. Parts of the results presented in this chapter are joint work with J. Christopher Beck and Wen-Yang Ku. Some results were previously published in one of the following papers:

1. Stefan Heinz and J. Christopher Beck, *Reconsidering mixed integer programming and MIP-based hybrids for scheduling*, in Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, N. Beldiceanu, N. Jussien, and E. Pinson, eds., Lectures Notes in Computer Science 7298, Springer, 2012, pp. 211–227.

2. Stefan Heinz Wen-Yang Ku and J. Christopher Beck, *Recent improvements using constraint integer programming for resource allocation and scheduling*, in Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, C. Gomes and M. Sellmann, eds., Lecture Notes in Computer Science 7874, Springer, 2013, pp. 12–27.

In Paper 1 we presented a first set of CIP models for the resource allocation and scheduling problem. We compared these models to existing MIP and CP formulations. In addition we conducted experiments using a logic-based Benders decomposition approach. The results indicated that the CIP models are promising. Further, even the MIP approach produced convincing results w.r.t. finding high-quality solutions. A disadvantage of this paper is that we did not use the best known relaxation for the LBBD approach. In Paper 2 we not only used a stronger sub-problem relaxation for the LBBD approach, but we also introduced a first version of an extended mixed-integer programming formulation which showed much better results than the previously used MIP model in the literature. Parts of the implementation were done by Wen-Yang Ku.

Outline. This chapter is organized as follows. In Section 5.1 we formally define the class of resource allocation and scheduling problems that are considered in this chapter. For each of the three approaches we discuss in this chapter, we present a computational study. The experimental setup which includes the computational environment and the considered test sets is presented in Section 5.2. We recall a logic-based Benders decomposition approach for this problem class in Section 5.3. We then take advantage of the linear relaxation developed in Section 2.3 for the cumulative constraint with optional jobs to present an improved LBBD approach. In Section 5.4 we present a mixed-integer programming approach. We first recall an existing model for this problem class and analyze its weaknesses. With this knowledge we develop an extended formulation for the resource allocation and scheduling problem. A CIP approach is presented in Section 5.5. A comparison of the three approaches is given in Section 5.6.

5.1 Problem definition

We study a basic allocation and scheduling problem which can be formalized as follows. Given a finite set of jobs $\mathcal{J} = \{1, \dots, n\}$ and a finite set of renewable resources $\mathcal{R} = \{1, \dots, m\}$, each job j must be assigned to exactly one resource k at an assignment cost c_{jk}. Each job j has a release date, R_j, and a due date, D_j, which define a time window within which the job has to be processed. Each resource $k \in \mathcal{R}$ has a bounded capacity $C_k \in \mathbb{N}$. Every job j has processing times $p_{jk} \in \mathbb{N}$ and resource demands $r_{jk} \in \mathbb{N}$ which both depend on the resource $k \in \mathcal{R}$. A feasible solution is an assignment where each job is placed on exactly one resource and a start time is assigned to each job such that no resource exceeds its capacity at any point in time. The goal is to find an optimal solution, that is, a feasible solution which minimizes the total resource assignment cost.

This problem class has been studied by a number of researchers over the past twenty years [Hoo00, JG01, Hoo05a, BP06, Tho01, Bec10, HB12a]. Mainly, LBBD approaches where presented and improved. In addition, a restricted version of the problem was considered where all jobs have a unit demand and all resources have a unit capacity. We showed in [HKB13] that instances of this restricted version are easily solvable with an LBBD, MIP, or CIP approach. Therefore, we do not consider this restriction in this dissertation.

5.2 Experimental setup

In this section we introduce the test instances which we use to analyze the performance of the different approaches for the resource allocation and scheduling problem. The computational environment for these experiments is described below.

5.2.1 Test sets

For our evaluation of the different models we are using the instances introduced in [Hoo04, Hoo05a, Hoo07]. In these papers four different test sets were introduced which we call TESTSETC, TESTSETE, TESTSETDE, and TESTSETDF. These test sets have different characteristics which we discuss in the following.

Test set testsetc. The test set TESTSETC was introduced in [Hoo04]. Here, all jobs have the same release and due date. The set contains 195 problem instances with the number of resources ranging from two to four and the number of jobs from 10 to 38 in steps of two. The maximum number of jobs for the instances with three and four resources is 32 while for two resources the maximum number of jobs is 38. For each problem size, we have five instances. The resource capacity is 10 and the job demands are generated with uniform probability on the integer interval $[1, 9]$.[1] See [Hoo04, Hoo05a] for further details w.r.t. generation of instances and the appendix of [HB12b] for further problem instance characteristics. These instances were also used in [Hoo07, Bec10, HB11, HB12a, HKB13, CCH13].

Test set testsete. As for test set TESTSETC, all jobs in TESTSETE have the same release and due date. These instances differ in the number of resources which range from 2 to 10. The number of jobs is fixed to five times the number of resources. In addition to these nine resource job combinations there is one which has 2 resources and 12 jobs. For each resource job combination five instances are available. These instances are again randomly generated. For more details we refer to [Hoo04, Hoo05a]. Overall this test set contains 50 instances. It was considered for example in [Hoo07, CCH13]

Test set testsetde. In [Hoo07] TESTSETDE was introduced. Here the release dates are the same whereas the due dates are different. All instance have 3 resources and differ in the number of jobs which need to be scheduled. The number of jobs range from 10 to 28 in steps of two. This gives ten resource job combinations. Each combination contains five randomly generated instances. For more details we refer to [Hoo07].

[1] In [Hoo04, Hoo05a] it is claimed that the demands are generated with uniform probability on the integer interval $[1, 10]$. Checking all 195 instances revealed that the demands 1 to 9 are uniformly distributed but the demand 10 never appears.

Table 5.1: Characteristics of the four test set we use for our experiments. Overall there are 335 instances.

test set	release dates	due dates	$\|\mathcal{R}\|$	$\|\mathcal{J}\|$	inst
TESTSETC	same	same	2,3,4	10, 12, . . . , 38	195
TESTSETE	same	same	2,3,. . . ,10	$5\|\mathcal{R}\|$	50
TESTSETDE	same	differ	3	10, 16, . . . , 28	50
TESTSETDF	differ	differ	3	14, 16, . . . , 28	40

Test set testsetdf. In [Hoo07] TESTSETDF was also introduced. Here, release and due dates vary. All instance have 3 resources and differ in the number of jobs which need to be scheduled. The number of jobs range from 14 to 28 in steps of two. This gives eight resource job combinations with each combination containing 5 randomly generated instances. For more details we refer to [Hoo07].

Table 5.1 summarizes the different characteristics of the four test sets. For each test set we state if the "release dates" of all jobs are the same or different. The same for the "due dates". Columns $|\mathcal{R}|$ and $|\mathcal{J}|$ states the range of the number resources and number of jobs, respectively. The last column "inst" show the number of instances which are part of the corresponding test set. Overall 335 instances are part of the evaluation.

5.2.2 Computational environment

Below we present different approaches for solving the previously introduced allocation and scheduling problems. All experiments are performed on Intel Xeon Core 3.20 GHz computers (in 64 bit mode) with 12 MB cache, running Linux, and 48 GB of main memory. For the different approaches we used different solvers. For the details about these, we refer to the corresponding sections.

5.3 Logic-based Benders decomposition

Logic-based Benders decomposition (LBBD) is a problem decomposition technique that generalizes Benders decomposition [Ben62, Geo72, Ben05]. It was first used in [HY95] and later formally defined in [HO03].

In this section we recall the idea of Benders decomposition and its generalization LBBD and restate a LBBD approach for the resource allocation and scheduling problem.

5.3.1 Background

The idea of Benders decomposition was introduced in 1962 to solve for example mixed-integer programs via decomposition [Ben62]. In that case, the decision variables are decomposed into two disjoint classes S_1 and S_2. This decomposition needs to satisfy that any assignment to the variables belonging to S_1 implies that for the remaining problem dual multipliers are defined. This is the case if the reduced problem, after fixing the variables belonging to S_1, is a continuous linear or nonlinear program. Hence, in case of a mixed-integer program, variables which need to be integral always belong to S_1. Having such a decomposition of the decision variables, the partitioning procedure works as follows. The *master problem*, consisting of the decision variables S_1 and constraints which are only defined via these variables, is solved to optimality. This problem is a relaxation of the original problem and so if the master problem is infeasible, the original problem is infeasible and the algorithm terminates. Otherwise, let $c^\star$ be the optimal value of the master problem. An optimal solution is used to initialize the *sub-problem*. All variables belonging to S_1 are fixed to the value they have in the optimal solution of the master problem. The dual problem of the resulting sub-problem (which is assumed to exist) is solved. If the optimal value of the dual sub-problem matches $c^\star$, it is proven that the current optimal solution of master problem (for the S_1 variables) and the dual sub-problem (for the S_2 variables) solve the original problem and the algorithm stops. Otherwise, the dual multipliers of the dual sub-problem are used to construct a so-called *Benders cut* which eliminates at least the used optimal solution from the master problem. Note that the Benders cut only contains variables which belong to S_1. This cut is added to the master problem and the procedure repeats until one of the two stopping criteria is reached, i.e., the master problem is infeasible or the optimal value of the master problem equals the optimal value of the sub-problem. This decomposition idea can use any partition of the original decision variables as long as the assumption w.r.t. the sub-problem is satisfied. Strong performance of this decomposition, however, is usually only achieved if the problem under investigation has a structure which suggests a decomposition of the variables (e.g., the matrix has block structure). For more details and a formal definition we refer to [Ben62, Geo72, Ben05].

The Benders decomposition approach requires that for the resulting sub-problem (after the primary variables are fixed) dual multipliers are defined. In [HY95, HO03] this restriction is relaxed. It is generalized to the assumption that the sub-problem can be *logically analyzed*: that it is possible to construct a Benders cut which removes the used optimal master solution from the master problem via a constraint. This generalization to logical analyzable sub-problems led to the name *Logic-based Benders decomposition*. In the next section we discuss how this approach can be applied to solve resource allocation and scheduling problems as has already been done in several papers [HO03, Hoo04, Hoo05b, Hoo05a, Hoo07, Bec10, CCH13].

5.3.2 Model

Logic-based Benders decomposition (LBBD) is a problem decomposition technique. Conceptually, some of the variables and constraints of a global problem model are removed, creating a master problem whose solution (in the case of minimization) forms a lower-bound on the globally optimal solution. The extracted problem components form one or more sub-problems where each sub-problem is an inference dual [HO03]. Based on a master problem solution, each sub-problem is solved, deriving the tightest bound on the master problem cost function that can be inferred from the current master problem solution and the constraints and variables of the sub-problem. If a bound produced by a sub-problem is not satisfied by the master problem, a *Benders cut* is introduced to the master problem. For global convergence, the cut must remove the current master problem solution from the feasibility space of the master problem without removing all globally optimal solutions. To achieve good algorithmic performance, the cut should remove a number of feasible master problem solutions that can be inferred to violate the bound of a sub-problem. For models where the sub-problems are feasibility problems, it is sufficient to solve the sub-problem to feasibility or generate a cut that removes the current master problem solution if the sub-problem is infeasible w.r.t. the master problem solution.

$$
\begin{aligned}
\min \sum_{k\in\mathcal{R}} \sum_{j\in\mathcal{J}} & c_{jk}\, x_{jk} \\
\text{s.t.} \sum_{k\in\mathcal{R}} & x_{jk} = 1 && \forall j \in \mathcal{J} && (5.1)\\
& \text{Sub-problem relaxation for resource } k && \forall k \in \mathcal{R} && (5.2)\\
& \sum_{j\in\mathcal{J}'} (1 - x_{jk}) \geq 1 && \forall k \in \mathcal{R}\ \forall \mathcal{J}' \in \mathcal{B}_k && (5.3)\\
& x_{jk} \in \{0,1\} && \forall j \in \mathcal{J}\ \forall k \in \mathcal{R}
\end{aligned}
$$

Model 5.1: Master problem model of a LBBD approach for the resource allocation and scheduled problem. Constraint (5.2) is a placeholder for a potential linear relaxation of the individual sub-problems. $\mathcal{B}_k \subseteq 2^{\mathcal{J}}$ is a subset of the power set of $\mathcal{J}$ representing the conflict sets (Benders cuts) for resource k.

$$
\begin{aligned}
& \texttt{cumulative}(\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}_{.k}, C_k) \\
& S_j \in \{R_j, \dots, D_j - p_{jk}\} && \forall j \in \mathcal{J}' \subseteq \mathcal{J}
\end{aligned}
$$

Model 5.2: Sub-problem for resource $k \in \mathcal{R}$. The index set $\mathcal{J}'$ contains those job indices which are assigned to resource k. The vectors $\boldsymbol{S}$, $\boldsymbol{p}$, and $\boldsymbol{r}_{.k}$ are restricted to $\mathcal{J}'$.

The resource allocation and scheduling problem which we investigate in this chapter suggests a natural decomposition. In a first step, jobs are assigned to resources and in a second step, jobs are scheduled. After the jobs are assigned to resources the remaining problem decomposes into several independent single-machine feasibility scheduling problems since there are no inter-job constraints. Therefore, the problem can be split into a master problem which is an assignment problem and sub-problems which are single cumulative scheduling problems. If all sub-problems are feasible, the assignment found by the master problem is valid for all resources and the corresponding cost is the global minimum. Otherwise, each infeasible sub-problem generates a Benders cut involving a set of jobs that cannot be feasibly scheduled.

Model 5.1 presents the master problem as an integer program. The binary decision variable x_{jk} is one if and only if job j is assigned to resource k. The objective function minimizes the total assignment cost. Equations (5.1) ensure that each job is placed on exactly one resource. Constraints (5.2) are a placeholder for a linear relaxation for each resource (sub-problem) which might be added to the master problem. Inequalities (5.3) present the Benders cuts which are added iteratively to the master problem. Model 5.2 shows a sub-problem. For each job $j \in \mathcal{J}' \subseteq \mathcal{J}$ which is placed on resource k, we have an integer start time variable S_j. The cumulative constraint ensures that the resource capacity C_k is not exceeded for any time step.

Experiments with LBBD models have shown that two aspects of the formulation are crucial for achieving good performance: the inclusion of a relaxation of each sub-problem in the master problem and a strong, but easily calculated Benders cut [Hoo07].

Sub-problem relaxation

The sub-problems are single-machine scheduling problems which are modeled each with a single cumulative constraint. The set of jobs which need be scheduled is given by a master problem solution. More generally, we have a single-machine scheduling problem with optional jobs where the job assignment is determine by a master problem solution. For such a structure we recalled two existing linear relaxations in Section 2.3 which we label as the *single* relaxation (Inequality (2.18)) and the *edge-finding* relaxation (Inequality (2.19)). In addition we showed that each energy-based propagation algorithm (see Definition 2.1) for the cumulative constraint implies a linear relaxation for the cumulative constraint with optional jobs. As a result of that we proposed the *energetic reasoning* relaxation (Inequality (2.23)). Each of these three relaxations can be added to the master problem. In Section 5.3.3 we present a computational study to show the impact of these linear relaxations.

Benders cut

Besides the sub-problem relaxation which is added to the master problem it is also important to create "good" Benders cuts. Given that in our case the sub-problems are feasibility problems without any visibility to the global optimization function, a Benders cut is a no-good constraint preventing the same set of jobs from being assigned to the resource again if the corresponding sub-problem is infeasible. Therefore, the cut will take the form:

$$\sum_{j \in \mathcal{J}'} (1 - x_{jk}) \geq 1$$

where $\mathcal{J}' \subseteq \mathcal{J}$ contains the indices of jobs which cannot be scheduled together on resource k. This constraint forces that at least one of the assigned jobs needs to be removed from the assignment for resource k. A strengthened cut can be produced by finding a subset of $\mathcal{J}'$ that also cannot be feasibly scheduled on resource k. Hooker [Hoo07] suggests a greedy procedure to find a minimal infeasible set by removing each job, one by one, from $\mathcal{J}'$ and resolving the sub-problem. If the sub-problem is still infeasible the corresponding job can be removed from the infeasible set, otherwise it stays in the set and the greedy procedure continues. In addition to that, we suggest a sorting step before this greedy procedure starts. Jobs are sorted w.r.t. their flexibility. A job which is fixed (meaning the start time variable is fixed) is not flexible w.r.t. its placement in the time dimension. On the hand a job which has a resource demand which matches the resource capacity is not flexible w.r.t. the resource capacity dimension. To measure this flexibility we compute for each job j and resource k the following value:

$$\text{flex}(k, j) = \frac{(D_j - R_j) \cdot C_k}{p_{jk} r_{jk}}.$$

The numerator gives the complete area where job j can be placed. This is divided by the energy of job j. A larger value indicates a more flexible job. In case of an infeasible sub-problem, all jobs $\mathcal{J}'$ are sorted in a non-increasing order w.r.t. this flexibility measure. This order defines, the order jobs are tested to be removable from $\mathcal{J}'$. Note if all jobs have the same release and due date, jobs with a smaller energy are checked first.

Table 5.2: Comparing different sub-problem relaxations added to the master problem.

				NORELAX				SINGLE				EDGE-FINDING				ENERGETIC			
test set	easy	hard	eval	opt	mast	sub	total	opt	mast	sub	total	opt	mast	sub	total	opt	mast	sub	total
TESTSETC	15	12	168	128	91.8	70.2	193.7	168	1.7	12.6	14.1	168	1.7	12.6	14.1	168	1.7	12.6	14.1
TESTSETE	7	0	43	38	55.5	16.8	71.9	43	9.9	2.0	11.4	43	9.9	2.0	11.3	43	9.9	2.0	11.2
TESTSETDE	26	0	24	24	4.6	5.8	9.0	24	2.4	4.0	5.6	24	2.1	3.3	4.8	24	1.8	3.1	4.3
TESTSETDF	18	0	22	22	0.7	3.1	3.5	22	0.8	2.9	3.4	22	0.7	1.9	2.5	22	0.8	1.7	2.4
all	66	12	257	212	55.0	39.2	101.1	257	2.7	8.6	11.6	257	2.7	8.3	11.3	257	2.7	8.3	11.2

5.3.3 Computational results

In this section we present computational results for the LBBD approach applied to the resource allocation and scheduling problem presented in Section 5.1. The goal is to analyze the importance of the sub-problem relaxation and the impact of the Benders cut strengthening technique. For these evaluations we use the four test sets introduced in Section 5.2.1: TESTSETC, TESTSETE, TESTSETDE, and TESTSETDF. Table 5.1 gives an overview of the different characteristics.

We realized the LBBD approach in the SCIP framework (version 3.0.1.5). That means that the master problem, which is a MIP, and the sub-problems, which are CPs, are solved with the same solver. For the master problem we use the default parameter settings of SCIP and SoPlex version 1.7.1 to solve the linear programming relaxations. For the sub-problems which are single cumulative constraints we run SCIP in CP mode.[2] Thereby, all techniques which are presented in Chapter 3 except the domain reduction stated in Theorem 3.30 are enabled. See Section 4.2.2 for more details. We also tried IBM ILOG CP Optimizer version 12.5 for solving the sub-problems. This leads, however, to a smaller number of solved instances and a slightly worse performance compared to the SCIP solver.

All experiments were executed in the computational environment described in Section 5.2.2. In addition we enforced a time limit of 2 hours for each instance.

Sub-problem relaxation

To analyze the importance of the sub-problem relaxation which is placed in the master problem, we perform four different experiments. In a first experiment we omit any relaxation, i.e., the master problem only consists of the assignment problem. This experiment is called NORELAX. Second, we add the basic single constraint knapsack relaxation (Inequality (2.18)) for each sub-problem. We refer to this experiment as SINGLE. Finally, we use the more involved relaxations which create for each sub-problem and any reasonable time window a linear knapsack constraint. Thereby, we distinguish between the relaxation which is based on the idea of the edge-finding propagation algorithm (Inequality (2.19)) and the energetic reasoning propagation algorithm (Inequality (2.23)). See Section 2.3 for more details. We refer to these experiments as EDGE-FINDING and ENERGETIC, respectively. In all four experiments the Benders cuts are strengthened with the greedy procedure described above.

Table 5.2 presents a summary of the computational results for the four test sets (TESTSETC, TESTSETE, TESTSETDF, and TESTSETDF). For each experiment we present the following

[2]In the interactive shell of SCIP, the CP solver settings can be set (and viewed) using the command `set emphasis cpsolver`. The method `SCIPsetEmphasis(scip, SCIP_PARAMEMPHASIS_CPSOLVER, TRUE)` does the same in the SCIP callable library.

information. The first column states the test set where “all” refers to all 335 instances. The columns “easy” and “hard” show the number of instances which are easy and hard. An instances is deemed to be easy if all four setups solve it in less than one second. An instance is assigned to the hard category if all four experimental conditions reached the time limit of 2 hours, i.e., none of the experiments can solve this instance. The instances which belong to one of these classes are removed from the evaluation. The column “eval” states the number of instances which are part of the evaluation. The sum of “easy”, “hard”, and “eval” instances equals the number of instances of the corresponding test set. For the instances which are part of the evaluation we present the number of instances which are solved to proven optimality (column “opt”) and the shifted geometric mean[3] with a shift of 10 seconds of the time spend in the master problem (column “mast”) and in the sub-problems (column “sub”). Finally, we state the overall running time in shifted geometric mean with the same shift in column “total”. All time measurements are given in seconds and for each instance we assume a minimum running time of 0.5 seconds. Instances which are part of the evaluation and hit the time limit are included with 7200 second in the calculation of the shifted geometric means.

In total there are only 12 instances which cannot be solved by any settings. These 12 hard instances belong to TESTSETC. All other 323 instances are solvable by those runs which include a sub-problem relaxation in the master problem. The setup which does not add any sub-problem relaxation to the master problem (NORELAX) only solves 278 instances (66 easy instances and 212 instances of the evaluation instances). Hence, all other setups solve 45 instances more that the NORELAX experiment. W.r.t. the overall performance the ENERGETIC relaxation is slightly better than the SINGLE constraint and EDGE-FINDING relaxations. This results from the test sets where jobs have different release or due dates: TESTSETDE and TESTSETDF. Recall that in TESTSETC and TESTSETE all jobs have the same release and due dates resulting in similar behavior for the settings SINGLE, EDGE-FINDING, and ENERGETIC.

The three experiments where we added a sub-problem relaxation to the master formulation outperform the version without any sub-problem relaxation (NORELAX) by a factor of almost 9 in shifted geometric mean. The latter needs 101.1 seconds in shifted geometric mean per instance whereas the SINGLE constraint relaxation, the EDGE-FINDING relaxation, and the ENERGETIC relaxation solve an instance in 11.6 seconds, 11.3 seconds, and 11.2 seconds, respectively. This is not surprising since in the NORELAX experiment the master problem “wildly” constructs assignments for the jobs in the beginning without any knowledge of the sub-problems. During the solving process, the Benders cuts eventually add information to the master problem which provide knowledge of the sub-problem structures.

In the case of the NORELAX experiment, most of the running time is spent in the master problem (except for the two smaller test sets TESTSETDE and TESTSETDF). One could conclude that the missing sub-problem relaxations in the master problem is the reason. This could imply several iterations between the master problem and the sub-problems to solve these problems. For the experiments where a sub-problem relaxation is provided for the master problem only the instances belonging to TESTSETE spend the larger amount of time in the master problem. We analyze this observation in more detail for the individual test sets: Tables 5.3–5.6 present detailed results for the four individual test sets. Each test set contains instances which differ in the number of available machines and the number of jobs which need to be placed. For each job-machine combination there are 5 instances.

[3]The shifted geometric mean of values $t_1, \ldots, t_n$ is $\left(\prod(t_i + s)\right)^{1/n} - s$, with shift s.

Table 5.3: Comparing different sub-problem relaxations which are added to the master problem w.r.t. the 195 instances of TESTSETC. Note that the job-machine combination with 4 machines and 10 jobs is omitted since all 5 instances belong to the easy category.

					NORELAX				SINGLE			
$\lvert\mathcal{R}\rvert$	$\lvert\mathcal{J}\rvert$	easy	hard	eval	opt	mast	sub	total	opt	mast	sub	total
2	10	1	0	4	4	0.5	1.4	1.5	4	0.5	0.5	0.6
	12	3	0	2	2	0.5	1.7	2.0	2	0.5	0.5	0.6
	14	2	0	3	3	0.5	2.0	2.2	3	0.5	0.5	0.5
	16	0	0	5	5	0.6	3.5	3.9	5	0.5	0.7	0.7
	18	0	0	5	5	3.7	12.0	15.6	5	0.5	1.6	1.6
	20	0	0	5	5	1.4	8.8	10.0	5	0.5	0.7	0.7
	22	0	0	5	5	25.3	69.5	104.6	5	0.5	17.2	17.3
	24	0	0	5	5	22.0	350.3	380.9	5	0.5	30.9	31.0
	26	0	0	5	4	771.8	1366.8	2546.8	5	0.5	30.3	30.4
	28	0	0	5	2	567.8	2441.3	5796.7	5	0.5	13.4	13.5
	30	0	1	4	0	–	–	–	4	0.5	623.1	623.1
	32	0	3	2	0	–	–	–	2	0.5	34.2	34.2
	34	0	2	3	0	–	–	–	3	0.5	242.0	242.0
	36	0	4	1	0	–	–	–	1	0.5	8.9	8.9
	38	0	1	4	0	–	–	–	4	0.5	93.3	93.3
3	10	2	0	3	3	0.5	1.1	1.2	3	0.5	0.5	0.6
	12	0	0	5	5	0.5	1.6	1.7	5	0.5	0.5	0.6
	14	0	0	5	5	0.9	3.8	4.7	5	0.5	0.6	0.7
	16	0	0	5	5	3.1	6.5	9.4	5	0.9	0.9	1.6
	18	0	0	5	5	14.7	14.7	29.0	5	3.0	1.5	4.2
	20	0	0	5	5	15.9	19.7	35.3	5	0.6	1.0	1.4
	22	0	0	5	5	38.4	32.0	70.5	5	0.7	1.4	1.9
	24	0	0	5	4	302.7	65.1	411.1	5	2.2	3.7	5.3
	26	0	0	5	5	625.0	127.8	793.7	5	1.3	35.4	36.9
	28	0	0	5	4	3381.7	440.7	3958.3	5	0.5	23.6	23.9
	30	0	0	5	0	–	–	–	5	4.0	174.9	178.7
	32	0	1	4	0	–	–	–	4	23.1	317.5	323.6
4	12	2	0	3	3	0.5	1.0	1.1	3	0.5	0.5	0.5
	14	0	0	5	5	0.7	2.3	2.8	5	0.6	0.7	1.0
	16	0	0	5	5	0.5	2.6	3.0	5	0.5	0.6	0.8
	18	0	0	5	5	4.0	8.7	12.5	5	0.9	0.9	1.7
	20	0	0	5	5	5.3	12.3	17.5	5	0.6	0.9	1.4
	22	0	0	5	5	46.5	25.7	72.8	5	1.1	1.0	1.9
	24	0	0	5	5	114.9	39.0	159.2	5	4.5	4.2	8.6
	26	0	0	5	3	1990.3	103.1	2148.7	5	6.3	6.3	12.1
	28	0	0	5	3	2113.9	152.6	2431.5	5	1.3	7.7	9.2
	30	0	0	5	2	3331.7	129.9	3513.0	5	12.9	22.7	49.8
	32	0	0	5	1	5186.9	202.9	5423.9	5	1.2	84.6	85.5
TESTSETC		15	12	168	128	91.8	70.2	193.7	168	1.7	12.6	14.1

The first two column "$|\mathcal{R}|$" and "$|\mathcal{J}|$" indicate this combination. For the five instances belonging to such a combination we state the same information as for the whole test set in Table 5.2. When a model did not solve any instances of a given size, we use '–' instead of 7200 for the running time. If all five instances of a job-machine combination are categorized as easy we omit this row in the table.

Test set testsetc. Table 5.3 presents the results for TESTSETC. Because all jobs in the test set have the same release and due date, the SINGLE knapsack constraint relaxation is equivalent to the EDGE-FINDING and ENERGETIC relaxation and all three experiments produce the same results. Therefore, we omit the results for the EDGE-FINDING and ENERGETIC relaxations in Table 5.3.

Independently of the sub-problem relaxation used, the performance gets better in terms

Table 5.4: Comparing different sub-problem relaxations which are added to the master problem w.r.t. the 50 instances of TESTSETE.

					NORELAX				SINGLE			
$\lvert\mathcal{R}\rvert$	$\lvert\mathcal{J}\rvert$	easy	hard	eval	opt	mast	sub	total	opt	mast	sub	total
2	10	3	0	2	2	0.5	1.0	1.1	2	0.5	0.5	0.6
2	12	3	0	2	2	0.6	2.6	3.0	2	0.5	0.8	0.9
3	15	1	0	4	4	0.8	2.3	2.8	4	0.5	0.7	0.9
4	20	0	0	5	5	6.1	5.2	10.0	5	2.1	1.3	2.9
5	25	0	0	5	5	1.6	5.8	7.1	5	0.5	0.8	0.9
6	30	0	0	5	5	2.7	6.4	8.8	5	0.8	1.0	1.7
7	35	0	0	5	5	43.4	19.5	66.4	5	5.7	2.3	8.1
8	40	0	0	5	4	239.6	34.5	292.4	5	42.5	3.7	46.6
9	45	0	0	5	2	2932.9	77.5	3065.8	5	98.2	4.8	104.0
10	50	0	0	5	4	1001.7	69.6	1103.0	5	18.3	3.3	21.4
TESTSETE		7	0	43	38	55.5	16.8	71.9	43	9.9	2.0	11.4

of the overall running time and number of solved instances if more machines are available. In case of NORELAX the approach fails completely if two or three machines are available and more than 28 jobs need to be placed, i.e., none of the 5 instances belonging to such a job-machine combination is solved within the time limit. In contrast, if four machines are provided the NORELAX experiment is able to solve instances with 32 jobs. When a relaxation is added we observe a much better performance. As for the NORELAX experiment, in case of two and three machines the model starts failing to solve larger instances (11 instances and 1 instance, respectively). In case of of four machines are available, all instances are solvable. Overall, the version (SINGLE) which places a sub-problem relaxation into the master problem is more than one order of magnitude faster than the version which starts without any sub-problem relaxation. The NORELAX experiment requires 193.7 seconds in shifted geometric mean whereas the SINGLE setup only needs 14.1 seconds.

As a first result, we can observe (as expected) that a sub-problem relaxation which is added to the master problem substantially improves the performance of this approach. Since all jobs have the same release and due date in this test set which of the three (SINGLE, EDGE-FINDING, and ENERGETIC) relaxations is used does not matter. The results for two machines indicate that the main issue when using a relaxation is to solve the sub-problems. Independently of the number of jobs, the time needed by the master problem can be neglected. This also holds for the 11 instances which are not solved. The reason for failing lies in the sub-problems. One of the sub-problems which needs to be solved consumes all the provided running time and 7 out of the 11 instances fail already in the first iteration. That means, the first assignment suggested by the master problem cannot be proved to be feasible or infeasible. This already shows the main disadvantage of the LBBD approach for this type of problem and was already observed and discussed [Bec10]. If the solution process gets stuck in one of the sub-problems the run returns without any feasible solution. For three and four machines the results also show that the time spend to solve the sub-problems dominates the time spend for solving the master problem. For both models, the master and the sub-problem, the time required increases with the number of jobs which need to be scheduled. However, the time needed for the sub-problems increases faster than the time consumed by the master problem.

Test set testsete. In TESTSETE all jobs have the same release and due date (as for TESTSETC) but the number of machines and jobs increase up to 10 and 50, respectively.

Table 5.5: Comparing different sub-problem relaxations which are added to the master problem w.r.t. the 50 instances of TESTSETDE. Note that the job-machine combination with 3 machines and 16 jobs is omitted since all 5 instances belong to the easy category.

					NORELAX				SINGLE				EDGE-FINDING				ENERGETIC			
$\lvert\mathcal{R}\rvert$	$\lvert\mathcal{J}\rvert$	easy	hard	eval	opt	mast	sub	total	opt	mast	sub	total	opt	mast	sub	total	opt	mast	sub	total
3	10	4	0	1	1	0.5	1.0	1.1	1	0.5	0.7	1.0	1	0.5	0.6	0.7	1	0.5	0.5	0.6
3	12	4	0	1	1	0.5	1.3	1.5	1	0.5	1.1	1.5	1	0.5	0.5	0.5	1	0.5	0.5	0.5
3	14	2	0	3	3	0.5	1.2	1.3	3	0.5	0.7	0.9	3	0.5	0.6	0.7	3	0.5	0.5	0.6
3	18	3	0	2	2	0.5	1.8	2.0	2	0.5	1.0	1.3	2	0.5	0.6	0.8	2	0.5	0.5	0.6
3	20	1	0	4	4	0.5	1.6	1.7	4	0.5	1.2	1.3	4	0.6	0.8	1.1	4	0.5	0.7	0.8
3	22	3	0	2	2	0.5	2.9	3.2	2	0.5	2.1	2.5	2	0.6	1.4	1.9	2	0.5	1.2	1.6
3	24	2	0	3	3	0.5	3.4	3.5	3	0.5	3.0	3.2	3	0.5	2.9	3.1	3	0.5	2.8	3.0
3	26	0	0	5	5	5.3	12.5	17.0	5	6.3	9.9	15.5	5	5.3	7.9	13.1	5	4.9	7.3	11.9
3	28	2	0	3	3	69.2	29.4	109.5	3	9.0	12.9	21.6	3	7.0	11.4	18.3	3	5.0	10.4	15.3
TESTSETDE		26	0	24	24	4.6	5.8	9.0	24	2.4	4.0	5.6	24	2.1	3.3	4.8	24	1.8	3.1	4.3

Table 5.4 presents the computational results. Again the results for the single constraint relaxation (SINGLE), EDGE-FINDING relaxation, and ENERGETIC relaxation are similar since all jobs have the same release and due date. Therefore, we restrict ourselves to the SINGLE and NORELAX results for the analysis.

The version which adds a sub-problem relaxation to the master problem (SINGLE) solves all 50 instances. In contrast, the version without a sub-problem relaxation fails on 5 instances. Overall the SINGLE version is a factor of more than 6 faster than the NORELAX version w.r.t. to the overall running time in the shifted geometric mean.

In the NORELAX condition most of the running time is spent in the master problem compared to time consumed by the sub-problems. More precisely, in the shifted geometric mean, 55.5 seconds are spent in the master problem whereas the sub-problems consume only 16.8 seconds. The same picture is observed for the SINGLE version. Here, however, the difference is not as large as for the NORELAX setup. Note that this is different to the results for TESTSETC, where the master problems are smaller w.r.t. to the number of variables and constraints.

Test set testsetde. The instances belonging to TESTSETDE all have three machines available and the number of jobs which need to be scheduled ranges from 10 to 28 in steps of two. The release dates are the same but the due dates differ. This implies that the EDGE-FINDING relaxation and ENERGETIC relaxation (potentially) differ from the SINGLE constraint relaxation. Table 5.5 states the summarized computational results.

All four setups (NORELAX, SINGLE, EDGE-FINDING, and ENERGETIC), even the one without any sub-problem relaxation in the master model, solve all 50 instances. The time spent in the master problem and the sub-problems increases in a similar fashion if more jobs need to be scheduled. The best performance is achieved by the experiment which adds the ENERGETIC relaxation to the master model. An instance is solved in 4.3 seconds in the shifted geometric mean. The EDGE-FINDING relaxation and SINGLE relaxation needs 4.8 seconds and 5.6 seconds, respectively. As expected the version without any sub-problem relaxation (NORELAX) is the slowest. It consumes 9.0 seconds in the shifted geometric mean. It is surprising that NORELAX is only a factor of 2 slower than the ENERGETIC relaxation.

Test set testsetdf. Finally, Table 5.6 states the results for TESTSETDF. Again all instances have three machines. In contrast to all other test set, the jobs differ in their release

Table 5.6: Comparing different sub-problem relaxations which are added to the master problem w.r.t. the 40 instances of TESTSETDF. Note that the job-machine combination with 3 machines and 16 jobs is omitted since all 5 instances belong to the easy category.

					NORELAX				SINGLE				EDGE-FINDING				ENERGETIC			
$\lvert\mathcal{R}\rvert$	$\lvert\mathcal{J}\rvert$	easy	hard	eval	opt	mast	sub	total	opt	mast	sub	total	opt	mast	sub	total	opt	mast	sub	total
3	14	4	0	1	1	0.5	1.2	1.3	1	0.5	1.1	1.1	1	0.5	0.9	1.1	1	0.5	0.9	1.1
3	18	2	0	3	3	0.5	2.0	2.3	3	0.5	1.8	2.0	3	0.6	1.4	1.8	3	0.6	1.2	1.6
3	20	2	0	3	3	0.6	2.0	2.4	3	0.5	1.6	1.9	3	0.5	1.4	1.7	3	0.5	1.3	1.6
3	22	0	0	5	5	0.5	2.3	2.5	5	0.6	2.3	2.6	5	0.6	1.6	2.0	5	0.7	1.5	1.9
3	24	1	0	4	4	0.7	5.1	5.8	4	1.0	4.8	5.8	4	1.4	3.6	4.9	4	1.5	3.4	4.8
3	26	3	0	2	2	2.1	10.1	11.9	2	3.3	9.9	12.7	2	1.2	2.4	3.7	2	1.2	1.9	3.2
3	28	1	0	4	4	0.5	1.8	1.9	4	0.5	1.7	1.8	4	0.5	1.5	1.7	4	0.6	1.5	1.8
TESTSETDF		18	0	22	22	0.7	3.1	3.5	22	0.8	2.9	3.4	22	0.7	1.9	2.5	22	0.8	1.7	2.4

and due dates. This potentially leads to a different sub-problem relaxation for the SINGLE constraint relaxation, EDGE-FINDING relaxation, and ENERGETIC relaxation. As for TESTSETDE, all setups are able to solve all 40 instances. For this test set there is almost no difference between the version without any relaxation (NORELAX) and the single constraint relaxation (SINGLE) indicating that the single constraint relaxation does not remove any feasible solutions of the pure assignment problem. The best performance again is achieved with the ENERGETIC relaxation which solves an instances in 2.4 seconds in shifted geometric mean. The other three experiments are slightly slower. The EDGE-FINDING relaxation requires 2.5 seconds, the SINGLE constraint relaxation needs 3.4 seconds, and the version without a relaxation consumes 3.5 seconds.

Summary. The results clearly indicate that it is important to add a sub-problem relaxation to the master problem. W.r.t. the instances under investigation it makes almost no difference if the single constraint relaxation (SINGLE) is used or one of the more involved relaxations (EDGE-FINDING or ENERGETIC). The overhead to construct these relaxations is negligible for these instances. The additional constraints in the master model do not lead to an increase in the time spend for solving it. Since the ENERGETIC relaxation dominates the EDGE-FINDING relaxation and single constraint relaxation w.r.t. the performance, it is reasonable to construct and use the ENERGETIC relaxation by default.

Benders cut strengthening

In the following we are focusing on the Benders cuts which are added to the master problem in case a sub-problem is infeasible. We perform two experiments. In one we construct the Benders cut directly from the suggested assignment of the master problem if the sub-problem is infeasible. In a second experiment we use the strengthening technique discussed in Section 5.3.2. We refer to these experiments as PURE and STRENGTHENED, respectively. In both cases we use the ENERGETIC relaxation for adding a linear sub-problem relaxation to the master problem. Table 5.7 present the results for the four test set TESTSETC, TESTSETE, TESTSETDE, and TESTSETDF. As before, for each test set we state the number of "easy" and "hard" instances as well as the number of instances which are part of the evaluation (column "eval"). For both experiments (PURE and STRENGTHENED) we give the number of instances solved to optimality (of the instances which are under evaluation) and three time measures. These are the running time spend in the master problem (column "mast"),

Table 5.7: Impact of the Benders cut strengthening technique.

				PURE				STRENGTHENED			
test set	easy	hard	eval	opt	mast	sub	total	opt	mast	sub	total
TESTSETC	74	10	111	109	7.0	25.8	38.1	109	2.3	27.8	31.4
TESTSETE	17	0	33	32	57.9	2.7	61.4	33	14.1	2.5	16.2
TESTSETDE	32	0	18	16	15.8	2.8	18.2	18	2.3	4.1	5.8
TESTSETDF	18	0	22	22	12.8	3.2	15.0	22	0.8	1.7	2.4
all	141	10	184	179	13.5	13.9	35.3	182	3.7	14.4	20.0

the sub-problems (column "sub"), and for solving the whole problem (column "total"). For the time measures we use the shifted geometric mean with a shift of 10 seconds.

First of all we want to note that the PURE setting is able to solve two instances which are not solvable with the STRENGTHENED setting. These two instances belong to TESTSETC and have 2 machines and 32 jobs. Having a closer look reveals that in the STRENGTHENED case the solver gets stuck in a sub-problem, i.e., the provided running time to solve this particular sub-problem is not enough to prove that the sub-problem is feasible or infeasible. Due to the different Benders cuts in case of the PURE experiment, the master problem produces different machine assignments. As a result the solver is able to evaluate the feasibility or infeasibility for the assignments suggested by the PURE version. Overall, however, the PURE setting solves 320 instances where the STRENGTHENED setup proves optimality for 323 instances.

The results summarized in Table 5.7 indicate that it is worthwhile to use the strengthening technique for the Benders cuts. Over all instances, we observe a speed-up of more than 50%. In case of the pure Benders cuts, an instances takes 35.3 seconds in the shifted geometric mean whereas the strengthened version only needs 20.0 seconds. For some of the individual test sets the speed ups are even larger. For the TESTSETDF, the results show a speed-up of a factor more than 6. Overall, we can say that the strengthening technique leads to consistently better results.

Summary

The results indicate that the LBBD approach is an effective method for the instances under investigation. Increasing the number of machines favors this approach since the problem decomposes into more sub-problems. Adding a sub-problem relaxation to the master problem leads to substantially better results. For all test instances we have almost a speed-up factor of 9 when comparing the version without any sub-problem relaxation to the one adding a sub-problem relaxation which is based on the energetic reasoning idea (see Table 5.2). Regarding the Benders cuts, it pays off to invest additional time to strengthen these cuts. For all instances we observe a speed-up of 50% when doing this (see Table 5.7). Figure 5.1 visualizes these aspects in a diagram. It shows the number of solved instances within a certain time for three LBBD setups. These are:

- ▷ The NORELAX experiment (Table 5.2) is shown as a dotted line (······). It adds no sub-problem relaxation to the master problem and does not strengthen the Benders cuts.
- ▷ The PURE experiment (Table 5.7) is printed as a dashed line (- - -). Here we add the energetic reasoning relaxation but do not strengthen the Benders cuts.

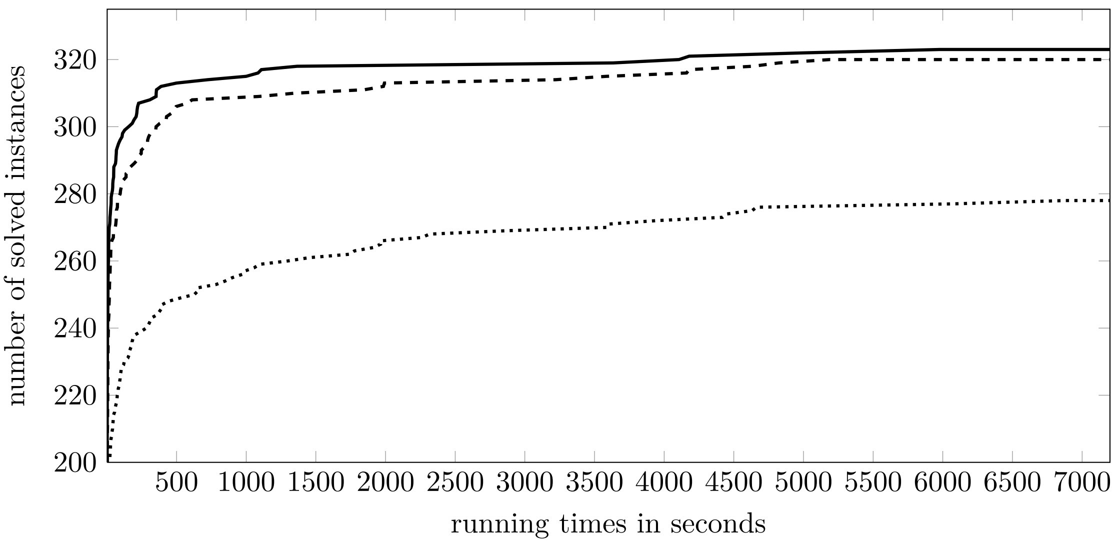


Figure 5.1: Diagram for the 335 resource allocation and scheduling instances and the three LBBD setups, showing the number of instances solved within a certain time. The version without a sub-problem relaxation and without Benders cut strengthening (NORELAX) is dotted (·····), the setting with ENERGETIC sub-problem relaxation and without Benders cut strengthening is dashed (- - -), and the LBBD setup with ENERGETIC sub-problem relaxation and Benders cut strengthening is solid (——).

- ▷ The ENERGETIC experiment (Table 5.2) and the STRENGTHENED experiment (Table 5.7) which are the same are depicted as a solid line (——). They use the energetic reasoning relaxation of each sub-problem in the master problem and the Benders cuts are strengthened.

The two versions which include a relaxation of the sub-problem in the master problem perform best. Strengthening the Benders cuts gives a small additional speed-up and results in few more instances being solved.

The experiments presented in this section also reveal one of the biggest disadvantages of LBBD. If one of the sub-problems is not solvable, the solution process gets stuck. In this particular case it returns without any feasible solution. We also observe that there are instances which are solvable with one setting and are not solved with another where the unsolvabilty is a result of this sub-problem issue. As suggested in [Bec10] it is of interest to generate several (different) optimal solutions for a particular master problem. If one optimal solution shows a weak performance w.r.t. to the sub-problems another optimal solution can be tried.

Overall, a sub-problems relaxation should be added to the master problem and it is worthwhile additional effort for strengthen the Benders cuts.

5.3.4 Conclusions

In this section, we discussed a LBBD approach for the resource allocation and scheduling problem which we analyze in this part of the dissertation. Such an approach was previously used in several others papers [HO03, Hoo04, Hoo05b, Hoo05a, Hoo07, Bec10, CCH13]. In the approach, the machine assignment problem is decoupled from the single machine scheduling problems. In a first phase (the master problem) a job-machine assignment is created. This assignment is checked in a second phase where sub-problems are solved to see if it is feasible for the individual machines. If the assignment is feasible for all machines, the approach terminates. Otherwise a Benders cut is created for each machine failing to schedule the assigned jobs. These cuts are added to the master problem and the extended master problem is solved to produce the next job-machine assignment candidate (see Section 5.3.2).

We discussed three different sub-problem relaxations which can be placed into the master problem (Section 5.3.2). Two of them (SINGLE and EDGE-FINDING) were previously known. We developed a sub-problem relaxation (ENERGETIC) which is based on the energetic reasoning propagation algorithm for the cumulative constraint. In case of the Benders cut created if a machine fails to schedule the assigned jobs, we recalled a strengthening idea for these cuts. The basic idea is to greedily remove jobs from the infeasible assignment as long as the shrunken assignment is still infeasible. We introduced a sorting of the jobs before the greedy approach is started (see Section 5.3.2) to retrieve stronger Benders cuts. Finally, we conduced an empirical study to analyze the importance of the sub-problem relaxations and Benders cut strengthening approach. The results strongly suggest to use our new developed ENERGETIC sub-problem relaxation together with the strengthened Benders cuts. In Section 5.6 we compare the best LBBD approach with a MIP and CIP method.

5.4 Mixed-integer programming

Despite the success of constraint programming for scheduling, the much wider penetration of mixed-integer programming (MIP) technology into business applications means that many practical scheduling problems are being addressed with MIP, at least as an initial approach. Furthermore, there has been impressive and well-documented improvements in the power of generic MIP solvers over the past decade [KAA$^+$11, AW13]. In this section we present a set of MIP models for the resource allocation and scheduling problem.

5.4.1 Background

The performance of MIP solvers depends on, besides other things, the chosen model. For the resource allocation and scheduling problem, which includes cumulative scheduling components, there exist several ways to model these structures with linear constraints: a time-indexed formulation which goes back to Pritskers et al. [PWW69, QS94], a flow-based formulation [AMR03], and an event-based formulation [TG96, Hoo07, KALM11]. The time-indexed models do not scale w.r.t. the time horizon whereas the flow-based and event-based formulations do. In practice, however, the time-indexed formulation gives much better results w.r.t. overall performance.

In this work we restrict ourselves to the time-indexed formulation. We recall a basic MIP model for the resource allocation and scheduling problem that is used in the literature to compare other approaches (like LBBD) against a MIP approach for these type of problems [HO03, Hoo04, Hoo05a, Hoo05b, HB12a, CCH13]. This basic model, however, is not tractable for modern MIP solvers which mainly rely on variable branching to implement the search. Discovering this drawback we introduced a second model which is an extended formulation [Bal05, VW10, CCZ13] of the first one, inspired by the master problem of the LBBD approach (see Section 5.3.2). We presented a variation of such an extended formulation in [HKB13].

In the next section, we present a standard model which is used in the literature for the allocation and scheduling problem. After discussing a disadvantage of this model for

$$
\begin{array}{lll}
\min & \displaystyle\sum_{k\in\mathcal{R}}\sum_{j\in\mathcal{J}}\sum_{t\in\mathcal{T}_{jk}} c_{jk}\, y_{jkt} & \\
\text{s.t.} & \displaystyle\sum_{k\in\mathcal{R}}\sum_{t\in\mathcal{T}_{jk}} y_{kjt} = 1 & \forall j\in\mathcal{J} \qquad (5.4)\\
& \displaystyle\sum_{j\in\mathcal{J}}\sum_{t'\in T_{jkt}} r_{jk}\, y_{jkt'} \le C_k & \forall k\in\mathcal{R}\ \forall t\in \bigcup_{j\in\mathcal{J}}\mathcal{T}_{jk} \qquad (5.5)\\
& y_{jkt}\in\{0,1\} & \forall j\in\mathcal{J}\ \forall k\in\mathcal{R}\ \forall t\in\mathcal{T}_{jk}
\end{array}
$$

Model 5.3: Basic integer programming model for the resource allocation and scheduling problem with $\mathcal{T}_{jk} = \{R_j, \dots, D_j - p_{jk}\}$ and $T_{jkt} = \{t - p_{jk} + 1, \dots, t\} \cap \mathcal{T}_{jk}$.

modern MIP solvers, we develop an extended formulation to partly overcome this particular disadvantage. In Section 5.4.3 we conduct a computational study which compares the two models. Doing this carefully indicates that our newly introduced model leads to a much better performance compared to the previously used model.

5.4.2 Models

In this section we first recall a MIP model which is frequently used in the literature for the resource allocation and scheduling problem to compare a MIP approach against other methods. We analyze this model and discuss how a disadvantage can be overcome by using the concept of extended formulations [Bal05, VW10, CCZ13] to reformulate this model.

Basic model

One way to model resource restrictions, as they arise in the resource allocation and scheduling problem, via linear constraints, is to use a *time-indexed* formulation [PWW69, QS94]. Thereby, for each time point, each job, and each resource a binary variable is introduced. Then sums over appropriate subsets of these variables control that resource capacity is respected at any point in time. This idea is used in several papers [HO03, Hoo04, Hoo05a, Hoo05b, HB12a, CCH13] for the resource allocation and scheduling problem to construct a MIP model. A binary decision variable, y_{jkt}, is created which is equal to 1 if and only if job j starts at time t on resource k. Model 5.3 utilizes these decision variables to realize a MIP model. Constraints (5.4) ensure that each job starts exactly once on one resource while Constraints (5.5) enforce the resource capacities on each resource at each time-point. The objective realizes the minimization of the assignment cost. This is a compact time-indexed formulation which has at most $|\mathcal{J}||\mathcal{R}||\mathcal{T}|$ variables and $|\mathcal{J}| + |\mathcal{R}||\mathcal{T}|$ constraints with $\mathcal{T} = \bigcup_{j\in\mathcal{J}}\bigcup_{k\in\mathcal{R}}\mathcal{T}_{jk}$. For a resource k and job j the set $\mathcal{T}_{jk}$ defines all feasible start points of job j on resource k.

State-of-the-art MIP solvers mainly search via variable branching. That means, that in our case, a binary variable is selected which has a fractional value in the corresponding linear programming relaxation solution. Two sub-problems are created. In one sub-problem this binary variable is fixed to zero in the other it is forced to be one. Such a branching leads to an unbalanced search tree for Model 5.3, independently of the chosen variable because the “one” branch, which decides that a certain job starts on a fixed resource at a certain time, implies many other binary variables must be zero via Constraints (5.4). This

$$\begin{aligned}
\min \sum_{k\in\mathcal{R}}\sum_{j\in\mathcal{J}} & c_{jk}\, x_{jk} && \\
\text{s.t.} \sum_{k\in\mathcal{R}} & x_{jk} = 1 && \forall j \in \mathcal{J} && (5.6)\\
\sum_{t\in\mathcal{T}_{jk}} & y_{kjt} = x_{jk} && \forall j \in \mathcal{J}\ \forall k \in \mathcal{R} && (5.7)\\
\sum_{j\in\mathcal{J}}\sum_{t'\in T_{jkt}} & r_{jk}\, y_{jkt'} \le C_k && \forall k \in \mathcal{R}\ \forall t \in \bigcup_{j\in\mathcal{J}} \mathcal{T}_{jk} && (5.8)\\
& x_{jk} \in \{0,1\} && \forall j \in \mathcal{J}\ \forall k \in \mathcal{R} &&\\
& y_{jkt} \in \{0,1\} && \forall j \in \mathcal{J}\ \forall k \in \mathcal{R}\ \forall t \in \mathcal{T}_{jk} &&
\end{aligned}$$

Model 5.4: An extended formulation of Model 5.3 for the resource allocation and scheduling problem with $\mathcal{T}_{jk} = \{R_j, \dots, D_j - p_{jk}\}$ and $T_{jkt} = \{t - p_{jk} + 1, \dots, t\} \cap \mathcal{T}_{jk}$.

decision reduces the remaining problem by one job. The “zero” branch, however, has in general no additional implications. It just removes one time point on one resource as a feasible start-time for a single job. This is a much smaller problem reduction compared to the “one” branch. The “one” branch also forces the job assignment cost to be accounted for in the LP relaxation whereas the “zero” branch in most cases leaves the dual bound unchanged. In the following section we discuss a way to (partly) resolve this branching issue.

An extended formulation

The basic model (Model 5.3) does not allow for a balanced variable branching. One way to resolve such an issue is to construct an extended formulation that introduces additional variables that allow the more precise modeling of certain structures of a problem. For example, extended formulations are often considered for a particular problem to create a formulation with polynomially many linear constraints that describes the convex hull of all feasible solutions. Such a formulation is called compact [CCZ13]. This concept is also used to handle and/or eliminate symmetry between different solutions or to provide variables which allow for better branching. The latter case is the one we want to achieve here. For more details about extended formulations we refer to [Bal05, VW10, CCZ13].

Inspired by the logic-based Benders decomposition approach for this problem, we introduce for each job and resource an additional binary variable. These variables allow us to model the job-machine assignment independently of the scheduling decision as it is done in the master problem of the LBBD method (see Model 5.1). Let x_{jk} be a binary decision variable which is one if and only if job j is assigned to resource k and zero otherwise. Note that such a decision completely ignores the time dimension of the problem. Having these decision variables and the y_{jkt} which are one if and only if job j is processed by resource k and starts at time-point t, Model 5.4 states an extended formulations for the basic model. As in the basic model, Constraints (5.8) ensure that the resource capacities are not exceeded at any point in time. The job-machine assignment is controlled by Equalities (5.6). These specify that each job is placed on exactly one machine. The newly introduced decision variables x_{jk} are linked via Constraints (5.7) to the y_{jkt} variables. This linking couples the job-machine assignment with the resource restrictions. The objective function minimizes the total job assignment cost. This model has $|\mathcal{J}||\mathcal{R}|$ additional variables and constraints compared to the basic Model 5.3.

The additional decision variables x_{jk} allow for a more balanced branching. Fixing such a decision variable to one introduces the same cost as fixing one of the corresponding y_{jkt} variables to one since the cost only depends on the job-machine assignment. On the other hand, fixing an x_{jk} for a job j and resource k to zero is equivalent to fixing several y_{jkt} to zero (see Constraints (5.7)). In addition, such a fixing potentially introduces an increase in the dual bound since it is clear that one of the other resources needs to handle this job. MIP solvers will most likely prefer to branch on the newly introduced decision variables because search heuristics [BGG+71] are based at least partially on the estimated increment in the dual bound.

Having the x_{jk} variables available, the sub-problem relaxations which we discussed in Section 5.3.2 for the LBBD approach can be added to the problem as well. The linear constraints of these energy-based linear relaxations (see Section 2.3.2), however, are redundant for the linear relaxation of Model (5.4). That is, a suitable linear combination of the linear constraints of Model 5.4 dominates each of the linear constraints of these relaxations. To prove this claim we first recall the definition of $e'_{jk}(a,b)$ for a non-empty interval $[a,b)$, a job j, and resource k:

$$e'_{jk}(a,b) = \max\{0, \min\{b-a, p_{jk}, \mathrm{ect}_{jk} - a, b - \mathrm{lst}_{jk}\}\} \cdot r_{jk}.$$

$e'_{jk}(a,b)$ computes the energy a job j contributes surely on resource k to the time window $[a,b)$ if it is assigned to this resource. Note that the earliest completion time ect_{jk} and the latest start time lst_{jk} of job j depend on resource k since the processing time is resource dependent. From Definition 2.1 it follows that $e'_{jk}(a,b)$ bounds the energy used by an energy-based propagation algorithm and therefore for an energy-based linear relaxation from above. Hence, it is sufficient to prove our claim for $e'_{jk}(a,b)$. We show that the linear constraints of the energetic reasoning relaxation, which use $e'_{jk}(a,b)$, are redundant. The energetic reasoning relaxation for a resource k is given as:

$$\sum_{j\in\mathcal{J}} e'_{jk}(a,b)\, x_{jk} \le C_k(b-a) \quad \forall (a,b) \in \{R_1,\dots,R_n\} \times \{D_1,\dots,D_n\} \,:\, a < b. \tag{5.9}$$

For more details about this relaxation and two others we refer to Section 2.3 and Section 5.3.2. To prove our claim, we show that for any resource k and non-empty interval $[a,b)$ the corresponding energetic reasoning linear constraint is dominated by the unit sum of Constraints (5.8) over the time points $t \in [a,b) \cap \mathbb{Z}$. The following theorem states that formally.

Theorem 5.1. For all resources $k \in \mathcal{R}$, and all pairs $(a,b) \in \{R_1,\dots,R_n\} \times \{D_1,\dots,D_n\}$ with $a < b$ it holds:

$$\sum_{j\in\mathcal{J}} e'_{jk}(a,b) x_{jk} \le \sum_{t=a}^{b-1} \sum_{j\in\mathcal{J}} \sum_{t'\in T_{jkt}} r_{jk} y_{jkt'} \le C_k(b-a).$$

Before we prove the theorem we prove the following lemma.

Lemma 5.2. For all jobs $j \in \mathcal{J}$, all resources $k \in \mathcal{R}$, and all pairs $(a,b) \in \{R_1,\dots,R_n\} \times \{D_1,\dots,D_n\}$ with $a < b$ it holds:

$$\sum_{t=a}^{b-1} \sum_{t'\in T_{jkt}} y_{jkt'} \ge \max\{0, \min\{b-a, p_{jk}, \mathrm{ect}_{jk} - a, b - \mathrm{lst}_{jk}\}\} \cdot x_{jk}. \tag{5.10}$$

Proof. We consider two cases. First, we assume $\mathrm{ect}_{jk} \leq a$ or $\mathrm{lst}_{jk} \geq b$. This implies that job j can potentially be processed completely before or after the time window $[a, b)$. Second, we assume $\mathrm{ect}_{jk} > a$ and $\mathrm{lst}_{jk} < b$. In this case job j overlaps with the time interval $[a, b)$ independently of the selected start point. Both cases capture all possibilities.

Case 1: If $\mathrm{ect}_{jk} \leq a$ ($\mathrm{lst}_{jk} \geq b$), job j can be processed completely before (after) the time interval $[a, b)$. In any case, the minimum at the right-hand side of Inequality (5.10) is non-positive since $\mathrm{ect}_{jk} - a \leq 0$ or $b - \mathrm{lst}_{jk} \leq 0$. Hence, the maximum evaluates to 0. The binary decision variables $y_{jkt'}$ are non-negative which implies that the left-hand side is greater than or equal to zero. This proofs that Inequality (5.10) holds if $\mathrm{ect}_{jk} \leq a$ or $\mathrm{lst}_{jk} \geq b$.

Case 2: If $\mathrm{ect}_{jk} > a$ and $\mathrm{lst}_{jk} < b$, job j contributes at least partly to the time window $[a, b)$. First, we evaluate the right-hand side of Inequality (5.10). Since $\mathrm{ect}_{jk} > a$ and $\mathrm{lst}_{jk} < b$, it follows that the minimum is strictly positive. Hence, the maximum computation can be ignored. This implies that the right-hand side evaluates to

$$\min\{b - a, p_{jk}, \mathrm{ect}_{jk} - a, b - \mathrm{lst}_{jk}\} \cdot x_{jk}.$$

Now we analyze for each $t \in \mathcal{T}_{jk}$ how often the binary decision variable y_{jkt} appears in the left-hand side of Inequality (5.10). This is equivalent to analyzing for each $t \in \mathcal{T}_{jk}$ which sets $T_{jkt'}$ with $t' \in [a, b) \cap \mathbb{Z}$ contain t. For each $t \in \mathcal{T}_{jk}$ it holds:

$$\{t\} \subseteq T_{jk\max\{a,t\}} \cap \cdots \cap T_{jk\min\{b-1,t+p_{jk}-1\}}.$$

Hence, t is part of at least $\min\{b-1, t+p_{jk}-1\} - \max\{a, t\} + 1$ many sets. This number can be bounded from below by taking the minimum over all possible combination of the subtraction:

$$\begin{aligned}
&\min\{b-1, t+p_{jk}-1\} - \max\{a, t\} + 1 \\
\geq\ &\min\{b-1-a+1, t+p_{jk}-1-t+1, t+p_{jk}-1-a+1, b-1-t+1\} \\
=\ &\min\{b-a, p_{jk}, t+p_{jk}-a, b-t\}.
\end{aligned}$$

This can be further bounded from below. We know that $t \in \mathcal{T}_{jk} = \{R_j, \ldots, D_j - p_{jk}\}$ which implies that $t \geq R_j = \mathrm{ect}_{jk} - p_{jk}$ and $t \leq D_j - p_{jk} = \mathrm{lst}_{jk}$:

$$\begin{aligned}
&\min\{b-a, p_{jk}, t+p_{jk}-a, b-t\} \\
\geq\ &\min\{b-a, p_{jk}, \mathrm{ect}_{jk} - a, b - \mathrm{lst}_{jk}\}.
\end{aligned}$$

Each binary decision variable y_{jkt} appears at least $\min\{b-a, p_{jk}, \mathrm{ect}_{jk} - a, b - \mathrm{lst}_{jk}\}$ times in the left-hand side of Inequality (5.10):

$$\sum_{t=a}^{b-1} \sum_{t' \in T_{jkt}} y_{jkt'} \geq \sum_{t \in \mathcal{T}_{jk}} \min\{b-a, p_{jk}, \mathrm{ect}_{jk} - a, b - \mathrm{lst}_{jk}\} \cdot y_{jkt}$$

Using Inequality (5.7) we substitute the sum of binary decision variables y_{jkt} with x_{jk}:

$$\sum_{t=a}^{b-1} \sum_{t' \in T_{jkt}} y_{jkt'} \geq \min\{b-a, p_{jk}, \mathrm{ect}_{jk} - a, b - \mathrm{lst}_{jk}\} \cdot x_{jk}.$$

This matches the right-hand side and proves that Inequality (5.10) holds for all job j with $\mathrm{ect}_{jk} > a$ or $\mathrm{lst}_{jk} < b$. □

Having this lemma, we prove Theorem 5.1.

Proof of Theorem 5.1. The theorem claim two inequalities:

$$\sum_{j\in\mathcal{J}} e'_{jk}(a,b)x_{jk} \overset{\overset{(1.)}{\downarrow}}{\leq} \sum_{t=a}^{b-1}\sum_{j\in\mathcal{J}}\sum_{t'\in T_{jkt}} r_{jk}y_{jkt'} \overset{\overset{(2.)}{\downarrow}}{\leq} C_k(b-a).$$

The second inequality follows from the unit sum of Constraints (5.8) over the time points $t \in \{a,\ldots,b-1\}$.

To prove the first inequality we use Lemma 5.2. Lemma 5.2 states that for all jobs $j \in \mathcal{J}$ it holds:

$$\sum_{t=a}^{b-1}\sum_{t'\in T_{jkt}} y_{jkt'} \geq \max\{0, \min\{b-a, p_{jk}, \mathrm{ect}_{jk} - a, b - \mathrm{lst}_{jk}\}\} \cdot x_{jk}.$$

The resource demands of each job are non-negative. Multiplying the above inequality with the resource demand r_{jk} of job j for resource k gives:

$$\begin{aligned}\sum_{t=a}^{b-1}\sum_{t'\in T_{jkt}} r_{jk}y_{jkt'} &\geq \max\{0, \min\{b-a, p_{jk}, \mathrm{ect}_{jk} - a, b - \mathrm{lst}_{jk}\}\} \cdot r_{jk} \cdot x_{jk}\\ &= e'_{jk}(a,b)\, x_{jk}.\end{aligned}$$

If we take the sum over all jobs and rearrange the summands, we get:

$$\sum_{t=a}^{b-1}\sum_{j\in\mathcal{J}}\sum_{t'\in T_{jkt}} r_{jk}y_{jkt'} \geq \sum_{j\in\mathcal{J}} e'_{jk}(a,b)x_{jk}.$$

This matches and proves the first inequality of Theorem 5.1. □

We proved that any linear constraint which is part of the energetic reasoning relaxation for cumulative constraint with optional jobs (Constraints (5.9)) is satisfied if the linear relaxation of Model 5.4 is satisfied. Since the energetic reasoning relaxation is at least as strong the edge-finding relaxation and single relaxation, it follows that none of these linear constraints belonging to one of these relaxations improve the linear relaxation of Model 5.4. Hence, they are redundant w.r.t. the linear relaxation.

Adding Constraints (5.9) to the model, however, might still have a positive effect. These constraints can lead to some domain reductions which would otherwise not happen. In the next section we analyze this empirically.

5.4.3 Computational results

In this section we compare the performance of different MIP models. We focus on the basic model (Model 5.3) and the extended formulation (Model 5.4). For the extended formulation we consider two models: one as is given by Model 5.4 and another for which we add the redundant Constraints (5.9). We refer to these models as BASIC, EXTENDED, and EXTENDED+(5.9), respectively. In our previous work [HKB13] we consider an extended formulation together with the redundant constraint provided by the edge-finding relaxation for each resource (see Constraints (2.19)). For the evaluation we are using the instances presented in Section 5.2.1. These are 335 instances in total which are divided into four test

Table 5.8: Comparing three MIP models using IBM ILOG Cplex with default parameter settings and allowing for a single thread.

				basic			extended			extended+(5.9)		
test set	easy	hard	eval	opt	nodes	time	opt	nodes	time	opt	nodes	time
testsetc	46	28	121	73	125092.5	557.2	77	99593.9	508.1	120	7565.5	70.0
testsete	14	0	36	23	81203.0	451.0	24	62959.1	412.5	36	8914.7	83.6
testsetde	27	0	23	23	1591.8	28.3	23	1907.4	25.0	23	656.0	11.8
testsetdf	21	0	19	19	123.0	1.8	19	136.5	1.9	19	83.2	1.5
all	108	28	199	138	38370.4	266.3	143	32712.7	245.0	198	4128.4	48.9

sets (testsetc, testsete, testsetdf, and testsetde). Each test set has a different characteristic w.r.t. the release dates and due dates of the jobs, the available resources, and the number of jobs which need to be scheduled. An overview of these characteristics can be found in Table 5.1.

For solving these MIP models we use IBM ILOG Cplex version 12.5.1.0. All experiments were executed in the computational environment described in Section 5.2.2. For each instance we enforced a time limit of 2 hours and allow for a single thread. The restriction to a single thread results from the fact that we are comparing these results to other approaches in Section 5.6 and the solver used for the other approaches is limited to a single thread. All other parameters are kept at their default values unless otherwise specified. In the end of this section, we briefly discuss the performance of the used solver running on 8 threads.

Preliminary results

Table 5.8 presents the results for the three models (basic, extended, and extended+(5.9)) and the four test sets. For each test set we state first the number of "easy" and "hard" instances. An instance is classified as "easy" if all three MIP models lead to a running time which is smaller than one second. If all three models fail to solve an instance, this instance is deemed to be "hard". Instances which belong to one of these two categories are removed from the evaluation. Therefore, column "eval" states the number of instances which take part in the evaluation. Hence, the sum of "easy" instances, "hard" instances, and "eval" instances equals the total number of instances belonging to the corresponding test set. For the instances which belong to the evaluation we state for each model the number of instances solved to proven optimality (column "opt") and the shifted geometric mean[4] for the number of visited search "nodes" and the overall running "time" in seconds. We applied a shift $s = 100$ for the number of search nodes and a shift $s = 10$ seconds for the overall running time. For the running time we assume a minimum running time of 0.5 seconds and instances which hit the time limit of 2 hours contribute with 7200 seconds to the shifted geometric mean to the overall running time if they are part of the evaluation set.

These results indicate that the additional decision variables lead to a small speed-up and 5 additionally solved instances. More precisely, the basic model needs 266.3 seconds in shifted geometric mean and solves 246 instances (108 easy and 138 instances which are under investigation) whereas the extended formulation (extended) requires 245.0 seconds and proves optimality for 251 instances. Adding the redundant constraints to the extended formulation (extended+(5.9)) results in a speed-up of factor larger than 5 and solves in

[4]The shifted geometric mean of values $t_1, \dots, t_n$ is $\left(\prod(t_i + s)\right)^{1/n} - s$, with shift s.

Table 5.9: Comparing three MIP models using IBM ILOG Cplex where any aggregation during the presolving phase is disabled and only a single thread is allowed.

				basic			extended			extended+(5.9)		
test set	easy	hard	eval	opt	nodes	time	opt	nodes	time	opt	nodes	time
testsetc	48	22	125	72	133257.7	615.5	116	10329.9	98.0	121	9404.0	102.3
testsete	14	0	36	23	81315.1	435.4	35	7301.8	72.7	36	6823.6	67.7
testsetde	28	0	22	22	1754.9	28.0	22	872.2	12.8	22	789.2	18.6
testsetdf	20	0	20	20	132.1	1.8	20	62.3	1.6	20	53.1	1.7
all	110	22	203	137	41009.8	284.0	193	4935.8	59.9	199	4527.7	62.6

total 306 instances. These are 55 instances more than the extended formulation and 60 instances compared to the basic model. Table 5.8 shows that these observations also holds for the four individual test sets.

Overall, these results indicate that the additional decision variables in the extended formulation (extended) result in a moderate speed-up. Adding constraints for which we proved that they do not improve the linear relaxation of the extended formulation, however, leads to a much better performance, which is surprising.

Detailed results

Analyzing the results more carefully reveals that the used solver removes the additional variables of the extended formulation (Model 5.4) during the presolving phase via aggregations. This transforms the problem back into basic model (Model 5.3). Therefore, the additionally added variables are not available for a more balanced branching during the tree search which was the main reason for introducing them. The differences in performance between these two models (basic and extended) mainly result from inferences being made via the additional variables before they are removed. This gives slightly smaller models (w.r.t. the number of variables) before the main search starts. Adding the redundant constraints to the extended formulation (extended+(5.9)) prevents the solver from deleting the additional decision variables. First, the solver does not discover that the Constraints (5.9) are redundant and as a consequence, the solver does not eliminate the additional variables. Due to this observation it is not clear if the additional decision variables or the redundant constraints are the reason for the observed speed-up. To stop the solver deleting these decision variables from the solving space we forbid any aggregation within the presolving phase[5] and rerun the experiments. Note this also prevents that any other aggregations from being made.

Table 5.9 presents the results where we forced that the newly introduced variables are not removed during the presolving phase. We state for each test set and model the same measures as in Table 5.8.

We verified that the additional variables are not removed via an aggregation from the search space. This allows for an interpretation w.r.t. the importance of the additional variables and the redundant Constraints (5.9). These results provide a different picture compared to the preliminary results and go along with our expectation.

Overall there are 22 instances which cannot be solved by any of the three experiments (22 hard instances), all in testsetc. Hence, 313 instances from 335 instances are solved within the time limit of 2 hours by at least one of the three models (basic, extended,

[5]To forbid any aggregation in IBM ILOG Cplex the parameters `CPX_PARAM_AGGIND` needs be set to 0. In the interactive shell that can be achieved via the command sequence "set preprocessing aggregator 0".

and EXTENDED+(5.9)). 110 instances belong to the easy category. These instances are solvable in less than one second by all three models. This gives in total of 203 instances which take part in the evaluation. All instances which are solved by the basic model are also solved by the two extended formulation. The largest number of instances solved by a single model is 309 (110 easy instances and 199 instances of the evaluation set) and is achieved by the extended formulation which contains the additional redundant constraints (EXTENDED+(5.9)). This model fails only on 26 instances. These are 4 instances more than the number of instances which belong to the hard category. Hence, these 4 instances are solved by the extended formulation (EXTENDED) and are not solved by the extended formulation with additional constraints (EXTENDED+(5.9)).

The basic model requires 284.0 seconds in shifted geometric mean and solves in total 247 instances (110 easy instances and 137 instances which take part in the evaluation). The extended formulation (EXTENDED) proves optimality for 303 instances and consumes 59.9 seconds in shifted geometric mean. This gives a speed-up of a factor of 4.7 and results in 56 instances being solved additionally. The number of search nodes is reduced by a factor of 8.3 between these two models. This indicates that the larger model (Model 5.4) takes approximately a factor of two more time for each search node. Adding the redundant constraints to the extended formulation (EXTENDED+(5.9)) does not give any additional speed-up but allows for solving 6 more instances.

Overall, the two extended formulations give significantly better results than the basic model for the resource allocation and scheduling problem. The performance improvement can be attributed to the additionally variables. The additional redundant constraints do not lead to a better run-time performance but allow for solving some additional instances.

In the following we analyze the four individual test sets (TESTSETC, TESTSETE, TESTSETDE, and TESTSETDF) in more detail. Tables 5.10–5.13 present, therefore, detailed results for the four test sets. Each test set contains instances which differ in the number of available machines and the number of jobs which need to be placed. For each job-machine combination there are 5 instances. The first two column "$|\mathcal{R}|$" and "$|\mathcal{J}|$" indicate this combination. For the five jobs belonging to such a combination we state the same information as for the whole test set in Table 5.9. That is the number of "easy" and "hard" instances, the number of instances which are part of the evaluation (column "eval"), and for each experiment the number of solved instances (column "opt"), the shifted geometric mean with a shift of 100 for the visited search "nodes", and the shifted geometric mean with a shift of 10 seconds for the overall running "time". For clarity, when a model did not solve any instances of a given size, we use '–' instead of 7200 for the running time. If all five instances of a job-machine combination are categorized as easy we omit the row in the table.

Test set testsetc. Table 5.10 states the results for TESTSETC. This test set has 195 instances with all jobs having the same release and due date. That implies that the number of additionally redundant Constraints (5.9) for the third experiment equals the number of available resources. Except in our previous work [HKB13] all other papers [HO03, Hoo04, Hoo05a, Hoo05b, HB12a, CCH13] consider only the basic model. The results shown here for this model go along with the results presented earlier by other authors.

Overall only 22 instances are categorized as hard. Thus, for 173 instances there exists at least one experiment which solves these instances. 48 instances are seen as easy, i.e., all three approaches solve these instances in less than one second. This leaves 125 instances for the evaluation.

Table 5.10: Comparing three MIP models w.r.t. the 195 instances of TESTSETC. Note that all job-machine combinations with 10 and 12 jobs and the combination with 4 machine and 14 jobs are omitted since all 5 instances belong to the easy category. Any aggregation during the presolving phase is disabled and only a single thread is allowed.

					BASIC			EXTENDED			EXTENDED+(5.9)		
$\lvert\mathcal{R}\rvert$	$\lvert\mathcal{J}\rvert$	easy	hard	eval	opt	nodes	time	opt	nodes	time	opt	nodes	time
2	14	4	0	1	1	14252.0	3.4	1	327.0	0.5	1	513.0	0.5
	16	1	0	4	4	2348.8	3.9	4	1255.9	3.5	4	911.8	2.6
	18	0	0	5	5	33614.4	82.4	5	1607.2	5.8	5	2071.7	13.6
	20	0	0	5	4	59719.3	214.2	5	1043.7	5.8	5	980.7	6.7
	22	0	0	5	2	275665.4	1532.7	5	69663.7	607.1	5	91281.4	949.9
	24	0	1	4	2	149758.6	806.4	3	31535.1	238.5	4	8459.2	126.3
	26	0	1	4	2	868228.8	3107.6	4	37391.1	225.2	3	73804.3	670.8
	28	0	1	4	3	267748.3	570.9	4	10541.5	78.1	4	11596.9	102.6
	30	0	2	3	2	379622.9	1099.9	3	10417.9	83.0	3	14406.2	84.9
	32	0	1	4	2	485380.2	2949.1	4	11817.9	151.6	4	13256.3	164.3
	34	0	1	4	1	313513.2	1831.8	4	19491.0	274.7	3	43893.2	825.4
	36	0	3	2	2	34373.3	195.8	2	13581.5	145.3	2	48522.2	424.9
	38	0	2	3	0	–	–	2	203291.2	3375.5	3	107382.9	2095.0
3	14	4	0	1	1	1728.0	1.4	1	50.0	0.8	1	105.0	1.0
	16	1	0	4	4	4660.9	8.0	4	1350.9	2.4	4	1595.8	2.1
	18	0	0	5	5	59837.7	123.2	5	1868.9	10.0	5	1800.6	10.8
	20	0	0	5	4	85848.2	197.0	5	3735.8	26.3	5	3655.6	42.1
	22	0	0	5	3	499757.4	1334.6	5	3167.9	25.0	5	7767.4	34.7
	24	0	0	5	3	376821.9	1555.7	5	14040.2	109.6	5	9940.4	102.7
	26	0	1	4	0	–	–	1	384894.2	2324.9	4	94079.4	944.1
	28	0	2	3	2	360939.4	1894.7	3	17664.4	161.4	3	4746.7	23.6
	30	0	0	5	0	–	–	4	76211.8	904.4	4	107766.9	1370.7
	32	0	2	3	0	–	–	3	72245.7	582.8	3	77707.9	730.4
4	16	3	0	2	2	1170.0	1.8	2	306.3	0.8	2	198.3	1.0
	18	0	0	5	5	2781.8	6.3	5	294.2	1.3	5	410.4	1.6
	20	0	0	5	5	32133.9	63.2	5	1752.3	6.5	5	1604.1	6.8
	22	0	0	5	5	24365.3	69.6	5	2573.1	12.9	5	1620.1	13.1
	24	0	1	4	2	629413.9	3559.2	4	12219.5	73.8	4	12114.6	105.0
	26	0	1	4	0	–	–	4	107933.8	577.8	4	65934.5	491.9
	28	0	0	5	1	503431.2	3331.2	5	48067.4	393.9	5	21687.0	284.1
	30	0	1	4	0	–	–	3	130007.7	1442.7	3	77287.9	1106.4
	32	0	2	3	0	–	–	1	165619.6	3245.8	3	85448.9	1225.5
TESTSETC		48	22	125	72	133257.7	615.5	116	10329.9	98.0	121	9404.0	102.3

The basic model (BASIC) solves 72 instances of the 125 instances under investigation. This gives a total of 120 instances including the 48 easy instances. For the basic model it is observed that the running time increases rapidly with the number of jobs which need to be scheduled. For 2 machines the basic model fails on all 5 instances which have 38 jobs. In case of 3 and 4 machines this already happens for 26 jobs.

The two extended formulations solve 116 and 121 instances of the 125 evaluation instances, respectively. Including the easy instances this gives 164 instances for the extended formulations EXTENDED and 169 instances for the extended formulation with the additional Constraints (5.9). The extended formulation EXTENDED is capable of solving at least one instance of each job-machine combination whereas EXTENDED+(5.9) solves at least 2 out of 5 instances for each job-machine combination. The overall running time increases if more jobs need to be scheduled (as expected) but the increase is much slower than for the basic model.

The basic model requires 615.5 seconds in shifted geometric mean to solve the instances under investigation. The two extended formulations need 98.0 seconds and 102.3 seconds, respectively. This gives a speed-up of more than a factor of 6 in shifted geometric mean. A similar picture can be observed for the number of search nodes. Here, the difference is almost a factor of 13.

Overall, the additional decision variables allow for a better performance of the used MIP

Table 5.11: Comparing three MIP models w.r.t. the 50 instances of TESTSETE. Note that the job-machine combinations with 2 machines and 10 and 12 jobs are omitted since all 5 instances belong to the easy category. Any aggregation during the presolving phase is disabled and only a single thread is allowed.

					BASIC			EXTENDED			EXTENDED+(5.9)		
$\lvert\mathcal{R}\rvert$	$\lvert\mathcal{J}\rvert$	easy	hard	eval	opt	nodes	time	opt	nodes	time	opt	nodes	time
3	15	3	0	2	2	865.5	2.0	2	189.9	0.7	2	80.6	0.8
4	20	0	0	5	5	7444.0	16.4	5	733.9	4.3	5	1199.9	4.9
5	25	1	0	4	4	16356.0	42.5	4	1981.0	6.1	4	2815.7	6.3
6	30	0	0	5	5	13642.4	44.5	5	1357.8	9.5	5	1603.5	10.0
7	35	0	0	5	3	311619.2	1539.3	5	12492.6	86.3	5	9346.5	80.6
8	40	0	0	5	1	551991.8	3294.5	5	23252.0	198.3	5	18124.0	169.2
9	45	0	0	5	0	–	–	5	80676.4	926.0	5	55259.3	699.6
10	50	0	0	5	3	444394.9	3437.6	4	57332.8	501.5	5	44726.2	484.1
TESTSETE		14	0	36	23	81315.1	435.4	35	7301.8	72.7	36	6823.6	67.7

solver IBM ILOG CPLEX. The additional redundant constraints do not give an additional speed-up but result in a few additionally solved instances. Both extended formulation seems to be more tractable for MIP solvers compared to the basic model.

Test set testsete. This set contains 50 instances. As for the instances of TESTSETC, all jobs have the same release and due date. This gives for each machine one additional (redundant) constraint for the third experiment. In this test set the number of machines and jobs increases up to 10 and 50, respectively. Table 5.11 states the detailed results.

Each instances can be solved by at least one model. Hence, the hard category contains zero instances. 14 instances belong to the easy category. This leaves 36 instances for the evaluation. The basic model solves 23 instances and fails on 13 instances. The extended formulation EXTENDED proves optimality for 35 of the 36 evaluation instances and hits the time limit on one of the largest instances. In total this model solves 12 instances more than the basic model. The extended formulation with additional redundant constraints is capable of solving all instances. W.r.t. the running time, the basic model is a factor of 6 slower than the two extended formulations. The basic model consumes 435.4 seconds in shifted geometric mean whereas the extended formulations need 72.7 seconds and 67.7 seconds, respectively. For this test set the shifted geometric mean for the extended formulation with additional redundant constraints is slightly smaller than for the extended formulation EXTENDED. This is not only a result of the additional solved instance as Table 5.11 indicates. Regarding the "exponential blow up", the results for all three models indicate that the running time increases if more jobs need to be scheduled until the we hit 50 jobs. All three models show a better performance for 50 jobs compared to 45 jobs. It is not clear why this is the case and needs to be analyzed in the future. Note that the same observation was made for the LBBD approach (see Table 5.4). For the basic model the increase factor is much larger than for the two extended formulations.

Overall, the additional variables are the main reason for the much better performance of the extended formulations. The additional constraints, however, give an additional small improvement.

Test set testsetde. In contrast to the two previous test sets, the instances of TESTSETDE have the same release date but differ in their due dates. Therefore, the additional redundant constraints used in the third experiment can be more than one per resource. In total this test set has 50 instances. Table 5.12 presents the results for these instances.

Table 5.12: Comparing three MIP models w.r.t. the 50 instances of TESTSETDE. Note that all job-machine combinations with less than 20 jobs are omitted since all 5 instances belong to the easy category. Any aggregation during the presolving phase is disabled and only a single thread is allowed.

					BASIC			EXTENDED			EXTENDED+(5.9)		
$\lvert\mathcal{R}\rvert$	$\lvert\mathcal{J}\rvert$	easy	hard	eval	opt	nodes	time	opt	nodes	time	opt	nodes	time
3	20	1	0	4	4	40.8	1.0	4	66.1	1.3	4	60.6	1.4
3	22	2	0	3	3	259.7	2.1	3	117.9	1.6	3	87.5	1.4
3	24	0	0	5	5	1344.4	21.0	5	822.9	6.8	5	1033.8	26.4
3	26	0	0	5	5	8151.4	54.8	5	1815.7	16.6	5	1033.2	20.3
3	28	0	0	5	5	11167.3	136.9	5	5139.1	60.0	5	5376.8	66.5
TESTSETDE		28	0	22	22	1754.9	28.0	22	872.2	12.8	22	789.2	18.6

Table 5.13: Comparing three MIP models w.r.t. the 40 instances of TESTSETDF. Note that all job-machine combinations with less than 20 jobs are omitted since all 5 instances belong to the easy category.

					BASIC			EXTENDED			EXTENDED+(5.9)		
$\lvert\mathcal{R}\rvert$	$\lvert\mathcal{J}\rvert$	easy	hard	eval	opt	nodes	time	opt	nodes	time	opt	nodes	time
3	20	4	0	1	1	1.0	1.0	1	1.0	1.0	1	1.0	1.1
3	22	0	0	5	5	82.8	1.1	5	49.0	1.2	5	38.8	1.1
3	24	0	0	5	5	244.7	1.3	5	127.6	1.2	5	119.9	1.4
3	26	1	0	4	4	44.6	1.2	4	9.2	1.4	4	36.2	1.5
3	28	0	0	5	5	242.5	3.8	5	90.6	2.5	5	40.4	3.1
TESTSETDF		20	0	20	20	132.1	1.8	20	62.3	1.6	20	53.1	1.7

All 50 instances can be solved within the 2 hours time limit by all three models. Therefore, the class of hard instances is empty. 28 instances (more than a half of the instances) belong to the easy category. That means, all three experiments solve these instances in less than one second. This leaves 22 instances which belong to the evaluation set. For these instances the basic model needs 28.0 seconds in shifted geometric mean. The two extended formulations use 12.8 seconds and 18.6 seconds, respectively. The EXTENDED model is, therefore, a factor of 2.2 faster than the basic model. The additional redundant constraint lead to a slow down compared to the EXTENDED experiment. This indicates that the additional constraints do not lead to important reductions if at all.

For this test set the additional variables x_{jk} give a speed-up of more than a factor of 2. The redundant constraints however additionally slow down the solving process. For all three experiment we can observe an increase in the running times if more jobs need to be scheduled.

Test set testsetdf. The last test set TESTSETDF contains 40 instances. These instances have different release and due dates resulting in potentially more than one redundant constraint for each resource in the third experiment. Table 5.13 summarizes the results for these instances.

20 instances of this test set are categorized as easy. This leaves 20 instances for the evaluation since all models are able to solve all 40 instances. All three experiments lead to similar results. The basic models requires 1.8 seconds in shifted geometric mean whereas the two extended formulations need 1.6 seconds and 1.7 seconds, respectively. For this test set the additional decision variables do not improve the overall performance. To generate results which allow for more interpretation, larger instances are required.

Table 5.14: Comparing three MIP models using IBM ILOG Cplex where any aggregation during the presolving phase is disabled and the number of available threads vary from one to eight.

				BASIC			EXTENDED			EXTENDED+(5.9)		
setting	easy	hard	eval	opt	nodes	time	opt	nodes	time	opt	nodes	time
1 thread	108	10	217	139	46616.4	330.3	195	6160.5	78.7	201	5676.5	82.0
8 threads	108	10	217	161	66985.0	214.4	206	8146.7	50.5	209	6357.0	44.0

Using parallel tree search

The conducted experiments restricted the solver to a single thread. This allows for a later comparison to other approaches (see Section 5.6). It has, however, been standard for the past 10 years for commercial MIP solvers to use multiple cores. Table 5.14, therefore, presents results where the solver is not restricted to a single thread. We reran our experiments and allowed the solver to take all eight cores available in our computational environment. As before we disabled all aggregations in the presolving phase to ensure that the additional decision variables are not removed. In Table 5.14 we compare the single thread and eight threads results for all three models. Thereby, the easy and hard instances are categorized w.r.t. all six experiments. The remaining columns present the same measures as in the previous tables in this section. The number of hard instances drops from 22 to 10 compared to the single thread results (see Table 5.9). Hence, for 325 of the 335 instances there exists at least one model which solves this instance with one or eight threads. The eight thread results are consistently better than the single thread results. For each model more instances are solved within the time limit and the overall running time drops. The observed speed-up for the different models is less than a factor of 2. The extended formulation with additional constraints (EXTENDED+(5.9)) leads to the best performance using eight threads. With this model the solver succeeds on 209 instances in 44.0 seconds (shifted geometric mean). Including the easy instances this amounts to 317 solved instances.

Summary

The computational study showed that the performance of a MIP approach heavily depends on the chosen model. Between the basic model (Model 5.3) which is widely used in the literature and the extended formulation (Model 5.4) there is a factor of almost 5 w.r.t. the overall running time in favor for the extended formulation (see Table 5.9). Figure 5.2 visualizes the performance of the three models BASIC, EXTENDED, and EXTENDED+(5.9). It states the number of solved instances within a certain time for each model. The BASIC model is depicted with a dotted line (······). The performance of the EXTENDED model is shown as dashed line (- - -). The results for the EXTENDED+(5.9) is given in a solid line (——). The two extended formulations lead to a much better performance of the chosen MIP solver. The extended formulation which contains the redundant constraints trails a little bit w.r.t. the performance in the beginning but eventually solves a few more instances than the extended formulation given in Model 5.4.

Until now we only analyzed the performance of the MIP models w.r.t. the time to solve the instances to proven optimality. MIP solvers are known to produce feasible solutions with a small optimality gap even in those cases where an instance is not solved. Analyzing the results of all instance 335 instances w.r.t. this aspect reveals that the largest gap for the basic model is 3.2% whereas for the two extended formulation a maximum gap of 2.3% is

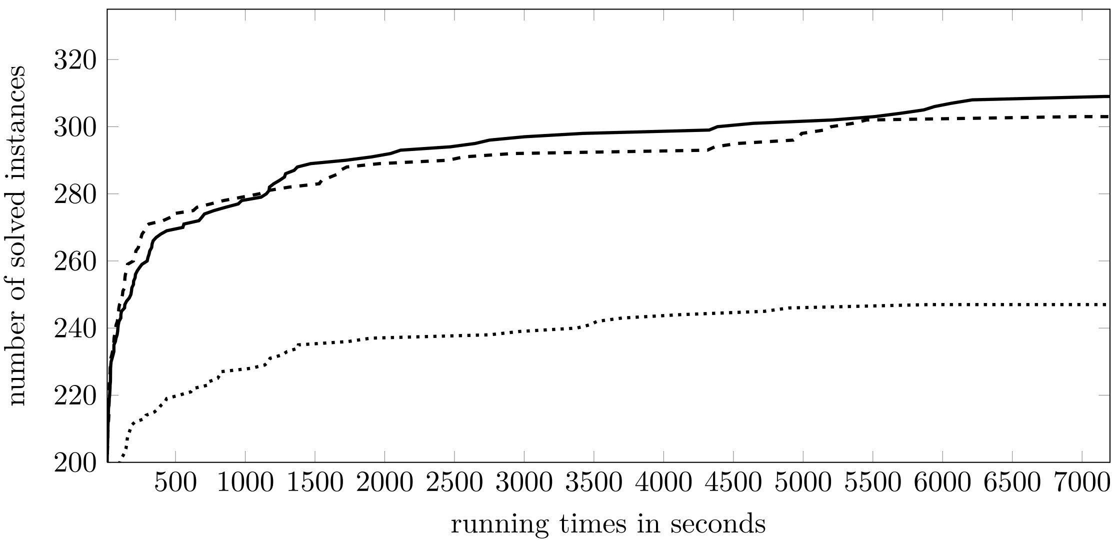


Figure 5.2: Diagram for the three MIP models on the 335 allocation and scheduling instances, showing the number of instances solved within a certain time. The BASIC model is dotted (······), the EXTENDED model is dashed (- - -), and EXTENDED+(5.9) model is solid (——).

observed. This shows that even for those instances which are not solved within the 2 hours time limit high-quality solutions are constructed. In Section 5.6, where we compare the MIP approach against a LBBD and a CIP approach, we take a closer look at the solution quality when an instance is not solved to proven optimality.

Overall, the extended formulation with the additional redundant constraints can be seen as the model to be chosen among the three analyzed models.

5.4.4 Conclusions

In this section we analyzed different MIP models for the resource allocation and scheduling problem. One is a basic formulation (Model 5.3) which is used in literature [HO03, Hoo04, Hoo05a, Hoo05b, HB12a, CCH13] for this problem. This compact model, however, is not tractable for modern MIP solvers which mainly perform variable branching (see Section 5.4.2). To overcome this issue we developed an extended formulation (Model 5.4).

In the previous section we presented two sets of experiments to empirically study the performance w.r.t. the different models. These experiments clearly showed that computational results have to be interpreted with care. More precisely computational results do not prove anything. They *only* give a hint or indicate that a certain hypothesis might be true. Our first series of experiments (Table 5.8), for example, could be misleading. Without a deeper analysis of these results, the shown numbers strongly indicate that the extended formulation is only useful if redundant constraints are added. The additional decision variables which we added to achieve a more balanced branching do not have much impact w.r.t. the performance. After a more careful analysis of the actual solver behavior we discovered that the solver converts the extended formulation into the basic model, removing the additional variables which are introduced to achieve a more balanced branching. In our second series of experiments, we ensured that the additional decision variables are not removed from the problem. This allows for an interpretation of the results w.r.t. the impact of the additional variables. We observed a speed-up of almost a factor of 5 for the extended formulation compared to the basic model (Table 5.9). Adding additional redundant constraints does not give any additional speed-up but does lead to a few more solved instances.

We demonstrated that the developed extended formulation is currently superior to the basic model w.r.t. the overall performance. The extended formulation with additional redundant constraints should be preferred when comparing other approaches for this problem against a MIP approach. We refer to Section 5.6 where such a comparison is presented.

5.5 Constraint integer programming

In the previous two sections, we discussed an LBBD and a MIP approach for the resource allocation and scheduling problem. The LBBD approach takes advantage of the problem structure and manually decomposes the problem into a master problem and several subproblem (see Sections 5.3). The MIP approach relies on the power of modern MIP solvers. Therefore, the complete problem is modeled as a MIP and a state-of-the-art solver is used. To some extent these two approaches can be seen as two extremes. The MIP approach relies on a black box solver whereas the LBBD approach uses a manual decomposition to solve this type of problems. The CIP approach we discuss in this section is in that sense a hybrid approach. We use a CIP solver which is capable of automatically taking advantage of the structure present in the problem without a manual decomposition.

5.5.1 Background

Constraint integer programming (CIP) was introduced by Achterberg in 2004 [Ach04, Ach07b, Ach09]. This paradigm builds a foundation for different problem classes, namely, mixed-integer programming (MIP), constraint programming (CP), and satisfiability testing (SAT). The basic concept is to combine solving techniques from the different fields and provide a solid framework for hybrid approaches. This hybridization allows for more structural approaches than linear constraints in MIP and clauses in SAT, combined with powerful low-level sophisticated solving techniques from MIP and SAT such as the linear programming relaxation and conflict analysis, respectively. Achterberg indicated the strength of the hybrid approach on a class of chip design verification instances [Ach07b, ABW07, Ach09]. It was shown that for certain problem classes the more structural CIP approach is superior to MIP and SAT methods. Furthermore, he indicated that this more general concept does not lose to much compared to pure MIP solvers for MIP instances. In this section we use this approach to tackle the resource allocation and scheduling problem.

5.5.2 Model

In this section we introduce the basic CIP model used to solve resource allocation and scheduling problems. This model is a concatenation of the master problem and the subproblems of the LBBD approach with the difference that we use the cumulative constraint with optional jobs instead of the cumulative constraint since the job resource assignments are not made a priori. Model 5.5 states the basic CIP model. For each job j and resource k we have a binary decision variables x_{jk} which is one if and only if job j is assigned to resource k and zero otherwise. Equation (5.11) ensures that each job is assigned to exactly one resource. In addition to the binary decision variables, for each job j and resource k, we have an integer start time variable S_{jk} which defines the starting point of job j on resource k if job j is assigned to resource k. The binary decision variables and the start time variables are used together with the cumulative constraint with optional jobs to ensure that the

$$
\begin{aligned}
\min \sum_{k\in\mathcal{R}} \sum_{j\in\mathcal{J}} & c_{jk}\, x_{jk} && \\
\text{s.t.} \sum_{k\in\mathcal{R}} & x_{jk} = 1 && \forall j \in \mathcal{J} && (5.11)\\
& \texttt{optcumulative}(\boldsymbol{S}_{\cdot k}, \boldsymbol{x}_{\cdot k}, \boldsymbol{p}_{\cdot k}, \boldsymbol{r}_{\cdot k}, C_k) && \forall k \in \mathcal{R} && (5.12)\\
& R_j \le S_{jk} \le D_j - p_{jk} && \forall j \in \mathcal{J},\ \forall k \in \mathcal{R} &&\\
& x_{jk} \in \{0,1\} && \forall j \in \mathcal{J},\ \forall k \in \mathcal{R} &&\\
& S_{jk} \in \mathbb{Z} && \forall j \in \mathcal{J},\ \forall k \in \mathcal{R} &&
\end{aligned}
$$

Model 5.5: Basic constraint integer programming model for the resources allocation and scheduling problem.

capacity of each resource is not exceeded, see Constraints (5.12). Having an individual start time variable for each job and resource, allows for inferences on the start time of a job depending on a potential assignment to the corresponding resource.

During the solving process the cumulative constraint with optional jobs contributes to the different algorithm components such as presolving, propagation, conflict analysis, linear relaxation, and primal heuristics. Therefore, all the techniques which are discussed in the previous chapters are utilized.

Presolving. Before the tree search starts, presolving detects and removes redundant constraints and variables. In case of the cumulative constraint with optional jobs, one can shrink the time windows of each job and remove *irrelevant* jobs from the scope of the constraint since this leads to potentially tighter linear relaxation (see Section 2.3). In particular, we use the dual reduction techniques developed in Chapter 3 that are able to remove redundant jobs from the cumulative constraint with optional jobs. Additionally, we can detect redundancy of a cumulative constraint with optional jobs. We refer to Section 3.5 for more details.

Propagation. Propagation (also known as node presolving) takes place in each search node and tries to infer bound changes from the branching decision. In case of the cumulative constraint with optional jobs we collect jobs which are assigned to a resource and apply the cumulative propagator (see Section 1.4.3). For the remaining jobs, we run singleton arc consistency to detect jobs which can no longer be feasibly scheduled and fix the corresponding binary decision variable to zero. In addition, if all resource assignment variables for a given resource are fixed, we try to solve the remaining individual cumulative constraint by itself, triggering a backtrack if no such solution exists. The same data structure used in presolving (see Chapter 3), can be used to perform this detection in a sound and general manner. In contrast to LBBD, these local sub-problems do not need to be solved. If a solution is found or the problem is proven infeasible, the global search space is reduced. However, if they are not solved, the main search continues.

Conflict analysis. If a search node is detected to be infeasible, conflict analysis can be used to analyze the infeasibility and retrieve conflict constraints. These conflict constraints restrict the remaining search space. For more details we refer to Section 2.2.1. In order to find restrictive conflict constraints it is important that for each inference made, an explanation algorithm exists which constructs an explanation for the inference (see Definitions 2.6). For the cumulative constraint we developed explanations for inferences made by energy-based propagation algorithms. We showed that these explanations can made available for the corresponding propagation algorithms for cumulative constraint with optional jobs. If an explanation is required during the analysis of an infeasible sub-problem, we use the explanations discussed in Section 2.2.3.

Linear relaxation. As for the LBBD and MIP approaches the cumulative constraint with optional jobs can provide a linear relaxation. We utilize the linear relaxations discussed in Section 2.3. In the following section we compare different linear relaxations w.r.t. their impact of solving resource allocation and scheduling problems.

Primal heuristic. Inspired by the clique structure of the problem (i.e., each job has to be assigned to one resource), we implemented a general purpose primal heuristic that assigns jobs to resources and solves the resulting decomposed scheduling problems. In MIP and CIP, a clique structure refers to a sets of binary variables that must sum up to at most one. This structure is easily detectable within a model and can be used within a fix-and-propagate heuristic. For more details we refer to [GBHW15].

5.5.3 Computational results

In this section we present a computational study to analyze the performance of the basic CIP model (Model 5.5) for the resource allocation and scheduling problem. We focus on the linear relaxation for the cumulative constraint with optional jobs which we discussed in Section 2.3. For the experiment we use the four test sets introduced in Section 5.2.1. These are TESTSETC, TESTSETE, TESTSETDE, and TESTSETDF. Table 5.1 gives an overview of the different characteristics of these sets.

We perform three experiments to analyze the importance of the linear relaxation for the cumulative constraint with optional jobs. In a first experiment, we omit a linear relaxation: the cumulative constraint with optional jobs does not place any linear constraint into the model. We call this experiment NORELAX. Second, we add the basic single constraint knapsack relaxation (Inequality (2.18)) for each cumulative constraint with optional jobs. We refer to this experiment as SINGLE. Finally, we use the more involved relaxation which creates for each cumulative constraint with optional jobs and any reasonable time window a linear knapsack constraint. Thereby, we consider the relaxation which is based on the idea the energetic reasoning propagation algorithm (Inequality (2.23)), see Section 2.3 for more details. We refer to these experiments as ENERGETIC. In all three experiments a time limit of 2 hours is enforced for each instance.

We realized the CIP approach in the SCIP framework (version 3.0.1.5). Therefore, we extended the constraint based framework with the required constraint handler for cumulative constraints with optional jobs. This constraint handler features all algorithmic components discussed for the CIP approach in this dissertation. The LP relaxations which need to be solved during the branch-and-bound algorithm, are solved with SoPlex version 1.7.1. All parameters are kept at their default values.

Table 5.15 presents the overall results for each test set w.r.t. to the three experiments (NORELAX, SINGLE, and ENERGETIC). The first column ("test set") states the test set name where the last line gives a summary for all instances of the four test sets. The following

Table 5.15: Comparing three CIP models using SCIP with default parameter settings.

				NORELAX			SINGLE			ENERGETIC		
test set	easy	hard	eval	opt	nodes	time	opt	nodes	time	opt	nodes	time
TESTSETC	20	17	158	142	5663.0	171.3	155	3876.6	36.8	153	5032.4	42.4
TESTSETE	9	2	39	39	2487.6	71.8	37	7525.4	73.6	36	11487.2	82.8
TESTSETDE	28	0	22	22	232.7	11.8	22	398.9	9.8	22	396.6	9.3
TESTSETDF	18	0	22	22	38.6	3.3	22	63.7	2.4	22	75.5	1.7
all	75	19	241	225	2676.6	93.5	236	2632.1	32.1	233	3376.4	35.8

three columns labeled with “easy”, “hard”, and “eval” show the number of instances which are categorized as easy and hard and the ones which are used to compute aggregated measures. An instance is classified as “easy” if all three experiments lead to a running time which is smaller than one second. If all three experiments fail to solve an instance, this instance is “hard”. Instances which belong to one of these two categories are removed form the evaluation. Column “eval” states the number of instances which take part in the evaluation. Hence, the sum of “easy” instances, “hard” instances, and “eval” instances equals the total number of instances belonging to the corresponding test set. For the instances which belong to the evaluation we state for each model the number of instances solved to proven optimality (column “opt”) and the shifted geometric mean[6] for the number of visited search “nodes” and the overall running “time” in seconds. We applied a shift $s = 100$ for the number of search nodes and a shift $s = 10$ seconds for the overall running time. For the running time we assume a minimum running time of 0.5 seconds and instances which hit the time limit of 2 hours contribute with 7200 seconds to the shifted geometric mean for the overall running time if they are part of the evaluation set.

Overall, only 19 instances are classified as hard where 17 belong to the largest test set TESTSETC and 2 to test set TESTSETE. 75 instances are easy since all three experiments solved these instances in less than a second. That leaves 241 instances for the evaluation. The experiment SINGLE which places a single knapsack constraint for each cumulative constraint with optional jobs into the linear relaxation solves 236 instances from the 241 base set. That are 3 more than the ENERGETIC experiment where each cumulative constraint with optional jobs puts a set of linear constraints into linear relaxation. The experiment which omits any linear relaxation for the cumulative constraint with optional jobs solves only 225 instances in total.

Adding a single knapsack constraint as relaxation for each cumulative constraint with optional jobs to the linear relaxation of the whole problem (SINGLE experiment) performs best w.r.t. running time. It takes 32.1 seconds in shifted geometric mean. Slightly slower is the ENERGETIC relaxation. This experiment needs 35.8 seconds in shifted geometric mean. Slower by almost a factor of 3 is the experiment which does not add any representation of the cumulative constraint with optional jobs to the linear relaxation. It requires 93.5 seconds.

Interestingly, the picture is different when analyzing the required search nodes. The NORELAX experiment and SINGLE experiment need 2676.6 and 2632.1 search nodes in shifted geometric mean, respectively. The ENERGETIC setup uses 3376.4 search nodes. This increase of the number of search nodes in case of the ENERGETIC experiment is reflected in the increase of the solving time compared to the SINGLE experiment. In case of the NORELAX experiment that is not the case. The NORELAX takes almost a factor of 3 more time compared to the SINGLE experiment but visits basically the same number of

[6]The shifted geometric mean of values $t_1, \ldots, t_n$ is $\left(\prod(t_i + s)\right)^{1/n} - s$, with shift s.

search nodes. Analyzing this carefully reveals that adding no relaxation for the cumulative constraint with optional jobs to the linear relaxation results in linear relaxation solutions which often satisfy all integrality conditions. That implies that this solution suggests a job resource assignment. In such a case each cumulative constraint with optional jobs needs to evaluate if the assigned jobs can be scheduled. This, however, takes time. In case we are adding a linear relaxation for each cumulative constraint with optional jobs to the linear relaxation, the resulting relaxation solutions are not that often integral. Many integral solutions that violate the resource constraints are now already forbidden in the relaxation so that this expensive check is not performed as often.

In the following we analyze the individual tests set in more detail. Therefore, Tables 5.16–5.19 present detailed results for the four individual test sets. Each test set contains instances which differ in the number of available machines and the number of jobs which need to be placed. For each job-machine combination there are 5 instances. The first two columns "$|\mathcal{R}|$" and "$|\mathcal{J}|$" indicate this combination. For the five jobs belonging to such a combination we state the same information as for the whole test set in Table 5.15. That is the number of "easy" and "hard" instances, the number of instances which are part of the evaluation (column "eval"), and for each experiment the number of solved instances (column "opt") and the shifted geometric mean with a shift of 100 for the search "nodes" and a shift 10 seconds for running "time". For clarity, when a model did not solve any instances of a given size, we use '–' instead of 7200 for the running time. On the other hand if all five instances of a job-machine combination are categorized as easy we omit this row in the table.

Test set testsetc. Test set TESTSETC is the largest test set with 195 instances. A characteristic of this test set is that all jobs have the same release and due date. In the presolving phase this changes. New lower bounds and upper bounds are imposed on the start time variables such that after the presolving phase it does not hold anymore that all jobs have the same release and due date. As a results we observe different behavior for the SINGLE experiment and ENERGETIC experiment due the different linear relaxation used. Table 5.16 presents the results for this test set in detail.

In total there are 17 instances classified as hard since all three experiments fail to solve them within the 2 hours time limit. 20 instances are easy since all three experiments solve these instances in less than a second. Hence, 158 instances out of 195 instances are part of the evaluation. 13 of the hard instances belong to the instances which have two machines, 3 belongs to the instances with 3 machines, and one instance to the instances with 4 machines.

The SINGLE experiment solves 155 instances. These are 2 more instances than for the ENERGETIC experiment which adds potentially several linear constraints to the linear relaxation for each cumulative constraints with optional jobs. The smallest number of solved instances is achieved by the experiment which does not add any constraint for the cumulative constraint with optional jobs to the linear relaxation. This experiment (NORELAX) solves 142 instances.

W.r.t. the running times, we observe the same order as for the number of solved instances. The SINGLE experiment performs best with an aggregated time of 36.8 seconds. A little bit slower is the ENERGETIC experiment which finishes with 42.4 seconds. The experiment which does not add any constraint to the linear relaxation (NORELAX) requires 171.3 seconds which is more than a factor 4 slower than the other two experiments.

In case of the number of search nodes the picture is slightly different. We still get the same order: SINGLE requires 3876.6 search nodes, ENERGETIC uses 5032.4 search nodes, and

Table 5.16: Comparing three CIP models (which differ in the used LP relaxation) w.r.t. the 195 instances of TESTSETC. Note that the job-machine combination with 4 machines and 10 jobs is omitted since all 5 instances belong to the easy category.

					NORELAX			SINGLE			ENERGETIC		
$\|\mathcal{R}\|$	$\|\mathcal{J}\|$	easy	hard	eval	opt	nodes	time	opt	nodes	time	opt	nodes	time
2	10	3	0	2	2	27.6	1.2	2	37.4	0.5	2	58.0	0.5
	12	3	0	2	2	40.9	2.2	2	33.9	1.0	2	40.6	1.0
	14	1	0	4	4	105.5	3.3	4	18.6	0.9	4	12.4	0.8
	16	0	0	5	5	299.7	5.7	5	159.9	1.5	5	160.3	1.3
	18	0	0	5	5	869.2	17.3	5	450.2	4.2	5	465.0	3.0
	20	0	0	5	5	771.1	23.7	5	277.4	2.1	5	272.2	2.1
	22	0	0	5	5	92583.3	492.8	5	68512.7	216.6	5	94927.7	248.8
	24	0	1	4	4	74525.5	492.0	4	41371.6	167.0	4	33916.9	145.3
	26	0	1	4	4	333090.6	1909.0	4	204310.6	551.1	4	242628.6	493.0
	28	0	1	4	4	62221.9	1677.4	4	4883.8	40.4	4	7090.8	58.9
	30	0	2	3	3	140165.3	3578.9	3	1382.7	25.0	2	9562.7	103.9
	32	0	3	2	0	–	–	2	168183.3	254.6	2	201784.1	240.7
	34	0	2	3	2	175617.6	3443.5	2	36844.8	349.6	2	48105.3	637.2
	36	0	2	3	0	–	–	3	31954.3	78.8	3	14253.1	67.7
	38	0	1	4	1	45858.3	4541.7	3	529645.4	732.2	4	467178.6	415.6
3	10	4	0	1	1	18.0	1.0	1	22.0	0.5	1	12.0	0.5
	12	1	0	4	4	45.9	1.7	4	52.7	0.7	4	84.6	0.6
	14	0	0	5	5	76.3	4.2	5	116.4	1.2	5	55.1	0.8
	16	0	0	5	5	338.7	11.2	5	626.1	2.8	5	601.6	2.6
	18	0	0	5	5	1212.6	32.7	5	2177.2	6.7	5	1995.7	5.6
	20	0	0	5	5	947.1	32.5	5	845.3	5.0	5	762.6	2.8
	22	0	0	5	5	2098.0	80.8	5	1556.5	9.0	5	2191.2	9.1
	24	0	0	5	5	16878.0	397.5	5	5635.8	23.0	5	8472.0	20.3
	26	0	0	5	5	43983.3	861.3	5	79194.2	180.9	5	63960.8	289.0
	28	0	0	5	5	83900.8	1415.0	5	20657.4	164.3	5	62750.3	493.5
	30	0	1	4	1	76426.6	5379.2	3	305148.5	867.7	3	376826.1	1108.7
	32	0	2	3	1	227123.4	6688.6	3	716991.9	2290.2	2	1252978.7	4006.7
4	12	3	0	2	2	17.9	1.0	2	4.0	0.5	2	5.9	0.5
	14	0	0	5	5	70.3	3.3	5	102.3	1.4	5	159.8	1.2
	16	0	0	5	5	86.7	4.2	5	158.9	2.0	5	227.4	1.9
	18	0	0	5	5	454.8	13.9	5	388.5	3.1	5	1067.1	4.0
	20	0	0	5	5	574.8	23.4	5	602.2	4.4	5	790.0	3.9
	22	0	0	5	5	3169.2	56.6	5	954.7	6.7	5	1966.4	6.8
	24	0	0	5	5	10305.4	168.4	5	11252.1	47.1	5	15894.9	53.4
	26	0	0	5	5	26088.6	374.5	5	16801.0	47.0	5	29678.3	71.5
	28	0	0	5	5	65873.1	767.6	5	6286.7	36.1	5	10232.7	50.7
	30	0	0	5	4	191297.6	2819.6	5	53458.5	264.6	5	84986.6	367.6
	32	0	1	4	3	621854.8	5599.4	4	179721.5	726.6	3	488455.4	980.4
TESTSETC		20	17	158	142	5663.0	171.3	155	3876.6	36.8	153	5032.4	42.4

NORELAX has 5663.0 search nodes in shifted geometric mean. The factors between these three experiments, however, are different compared to the factors between the running times. The increase in time and search nodes from the SINGLE experiment to the ENERGETIC experiment is basically the same factor, but from the SINGLE experiment to the NORELAX experiment we only have a factor slightly smaller than 1.5 for the search nodes whereas we have a factor larger than 4 for the running time. As pointed out in the previous section, the reason for that comes from the overhead of checking potential solutions for a single cumulative constraint with optional jobs.

For all three experiments, the number of search nodes needed and the running time increases (as expected) if more jobs need to be scheduled. This is independent of the available number of machines. The number of instances which are categorized as hard, however, decreases with the number of available machines.

Overall, the SINGLE experiment gives the best results. It solves the highest number of instances and requires the smallest running time.

Table 5.17: Comparing three CIP models (which differ in the used LP relaxation) w.r.t. the 50 instances of TESTSETE.

					NORELAX			SINGLE			ENERGETIC		
$\lvert\mathcal{R}\rvert$	$\lvert\mathcal{J}\rvert$	easy	hard	eval	opt	nodes	time	opt	nodes	time	opt	nodes	time
2	12	3	0	2	2	81.9	4.0	2	80.7	0.8	2	136.7	0.7
3	15	1	0	4	4	84.6	4.6	4	182.8	1.6	4	278.9	2.0
4	20	0	0	5	5	276.9	10.2	5	692.0	6.5	5	1134.7	7.4
5	25	0	0	5	5	215.9	12.5	5	273.6	3.6	5	459.0	4.7
6	30	0	0	5	5	571.7	20.0	5	2068.1	11.7	5	3936.5	13.7
7	35	0	0	5	5	4936.0	87.7	5	28192.3	100.4	5	39081.2	113.4
8	40	0	0	5	5	24030.1	348.7	4	191942.9	821.0	4	231303.8	768.5
9	45	0	1	4	4	123934.2	1432.1	3	762579.6	3124.3	3	901795.3	3460.5
10	50	0	1	4	4	63238.7	793.0	4	162859.1	695.4	3	446625.0	1218.0
TESTSETE		9	2	39	39	2487.6	71.8	37	7525.4	73.6	36	11487.2	82.8

Test set testsete. Table 5.17 states the detailed results for test set TESTSETE. This test set contains 50 instances and has the same property concerning release and due dates of the jobs as TESTSETC: all jobs have the same release and due dates. The characteristic of this test set, however, differs for the number of available machines and jobs, which increases up to 10 and 50, respectively.

Even though all jobs have the same release and due dates the result for the SINGLE experiment differs from the result for the ENERGETIC experiment. This is the case since presolving infers lower and upper bounds for the start time variables. As a result the presolved model has different release and due dates for the jobs. Since these dates are used to create the linear relaxation for the cumulative constraint with optional jobs, the two experiments have different relaxations.

In total there are 9 instances classified as easy and 2 instances as hard. That leaves 39 instances for the evaluation. The 2 hard instances belong to the two largest instance classes with 9 and 10 machines.

Surprisingly, the best performance w.r.t. the overall running time is given by the NORELAX experiment. It solve 39 instances with an aggregated time of 71.8 seconds. NORELAX solves 2 instances more than the SINGLE experiment which takes 73.8 seconds and 3 instances more than the ENERGETIC experiment which requires 82.6 seconds. Analyzing this more carefully reveals that the available primal heuristics perform much better in the NORELAX experiment compared to other two experiments. For these instances it seem to be beneficial if good primal solutions are present early in the search as more search nodes can then be pruned before processing.

As before, the time spent per search node is much larger for the NORELAX experiment compared to the two other experiment. In total the NORELAX experiment uses 2487.6 search nodes in shifted geometric mean. This number is increased by a factor larger than three in the SINGLE experiment while it takes only slightly more running time than NORELAX. The ENERGETIC setup needs a factor of 4.6 more search nodes compared to the NORELAX experiment.

Until the second largest problem class (9 machines and 45 jobs) the required search nodes and the running time increase as expected for all three experiments when the problems get larger. For the largest problem class (10 machines and 50 jobs), however, we observe that the number of search nodes and the running time are much smaller than for the second largest problem class. It is not clear why this is the case and needs to be analyzed in the future. Note that the same observation was made for LBBD and MIP approach, see Table 5.4 and see Table 5.11, respectively.

Table 5.18: Comparing three CIP models (which differ in the used LP relaxation) w.r.t. the 50 instances of TESTSETDE. Note that the job-machine combination with 3 machines and 10 and 12 jobs is omitted since all 5 instances belong to the easy category.

					NORELAX			SINGLE			ENERGETIC		
$\lvert\mathcal{R}\rvert$	$\lvert\mathcal{J}\rvert$	easy	hard	eval	opt	nodes	time	opt	nodes	time	opt	nodes	time
3	14	3	0	2	2	13.0	0.7	2	23.4	0.9	2	29.9	0.8
3	16	4	0	1	1	18.0	1.7	1	47.0	1.4	1	23.0	0.8
3	18	3	0	2	2	67.7	2.4	2	105.8	2.7	2	152.7	1.6
3	20	1	0	4	4	44.3	2.7	4	88.7	2.0	4	105.4	1.5
3	22	3	0	2	2	37.1	3.7	2	191.8	5.0	2	169.2	3.6
3	24	2	0	3	3	54.4	7.9	3	185.7	8.5	3	93.3	17.5
3	26	0	0	5	5	648.7	28.3	5	1085.1	18.7	5	1612.4	17.9
3	28	2	0	3	3	4578.6	75.1	3	7309.3	53.9	3	4729.9	36.9
TESTSETDE		28	0	22	22	232.7	11.8	22	398.9	9.8	22	396.6	9.3

Overall, the NORELAX experiment and the SINGLE experiment show the best performance for this test set. For larger instances the NORELAX experiment seems to be slightly superior to the SINGLE experiment. One reason for the strong performance of the NORELAX experiment is the poor performance of the primal heuristics in case we add linear constraints to the linear relaxation of the problem for each cumulative constraint with optional jobs (as we do for the SINGLE experiment and ENERGETIC experiment). This should be further analyzed and overcome in the future.

Test set testsetde. Test set TESTSETDE contains 50 instances which all have three machines available and the number of jobs range from 14 to 28 in steps of two. The release dates for all jobs are the same, but the due dates differ. As a result we get different linear relaxations with the experiments SINGLE and ENERGETIC. Table 5.18 shows the computational results.

All three experiments are able to solve all 50 instances to optimality. Hence, none of the instances is categorized as hard. 28 instances are categorized as easy since all three experiments can solve them in less than one second. The remaining 22 instances are part of the evaluation.

The best performance w.r.t. time is achieved by the ENERGETIC experiment with a shifted geometric mean of 9.3 seconds, closely followed by the SINGLE experiment with 9.8 seconds. Slowest is the NORELAX experiment which takes 11.8 second in shifted geometric mean. In case of the search nodes the picture is different again. The NORELAX experiment which takes the most time requires the smallest number of nodes. It needs 232.7 search nodes in shifted geometric mean. The other two experiments, SINGLE and ENERGETIC, visit 398.9 and 396.6 search nodes, respectively. Hence, the NORELAX experiment spends much more time per node than the SINGLE and ENERGETIC experiment.

If the number of jobs increases, the required search nodes and the running time increase as well. The increase in time is, however, moderate. For the larger job-machine combination all three experiment are well below 100 seconds in shifted geometric mean.

Overall, the best performance is achieved by the two experiments which add a relaxation for each cumulative constraint with optional jobs to the linear relaxation. Between these two experiments neither can be declared the clear winner.

Test set testsetdf. The final test set TESTSETDF contains 40 instances. All instances have three machines available and the jobs range from 14 to 28 in steps of 2. These instances

Table 5.19: Comparing three CIP models (which differ in the used LP relaxation) w.r.t. the 40 instances of TESTSETDF. Note that the job-machine combination with 3 machines and 14 and 16 jobs is omitted since all 5 instances belong to the easy category.

					NORELAX			SINGLE			ENERGETIC		
$\vert\mathcal{R}\vert$	$\vert\mathcal{J}\vert$	easy	hard	eval	opt	nodes	time	opt	nodes	time	opt	nodes	time
3	18	2	0	3	3	30.2	2.4	3	52.1	1.8	3	69.4	1.6
3	20	3	0	2	2	20.5	1.2	2	32.8	1.1	2	44.0	0.9
3	22	0	0	5	5	30.6	2.5	5	53.5	2.0	5	60.8	1.2
3	24	1	0	4	4	66.9	6.9	4	112.7	4.9	4	159.5	3.2
3	26	2	0	3	3	50.2	3.4	3	77.2	2.0	3	51.1	0.9
3	28	0	0	5	5	32.4	3.2	5	53.2	2.2	5	69.6	2.1
TESTSETDF		18	0	22	22	38.6	3.3	22	63.7	2.4	22	75.5	1.7

have different release and due dates. This results in different linear relaxations used in the SINGLE and ENERGETIC experiments. Table 5.19 summarizes the results for these instances.

Each experiment solves all 40 instances to optimality. Hence, none of the instances belong to the hard category. 18 instances are categorized as easy since each setup can solve them in less than one second. That leaves 22 instances for the evaluation.

The best performance w.r.t. the running time is achieved by the ENERGETIC experiment which adds several linear constraints to the linear relaxation for each cumulative constraint with optional jobs. It requires 1.7 seconds in shifted geometric mean. The SINGLE and NORELAX experiment need 2.4 and 3.3 seconds, respectively.

W.r.t. the search nodes we get the same picture as before. The NORELAX experiment requires many fewer search nodes compared to the other two experiments. The NORELAX experiment processes 38.6 search nodes in shifted geometric mean whereas the SINGLE and ENERGETIC experiments visit 63.7 and 75.5 search nodes, respectively.

For this test set we cannot observe that the running time and the number of search nodes increase if more jobs need to be placed. All job-machine combination result in similar performance numbers. Except for the job-machine combination where 24 jobs need to be scheduled, which needs more running time and more search nodes in all three experiments.

Overall, all three experiments deliver a good performance on this test set.

5.5.4 Conclusions

In this section we compared three different linear relaxations for the cumulative constraints with optional jobs used within a CIP model for the resource allocation and scheduling problem. The computational study indicates that adding a linear relaxation of these constraints to the linear relaxation of the problem pays off. For almost all test sets (except TESTSETDE) more or the same number of instances are solved within the time limit. Figure 5.3 visualizes the performance of the three experiments NORELAX, SINGLE, and ENERGETIC. It states the number of solved instances within a certain time for each experiment. The NORELAX experiment is depicted with a dotted line (······). The performance of the SINGLE experiment is shown as dashed line (- - -). The results for the ENERGETIC experiment is given in a solid line (——). The two experiments SINGLE and ENERGETIC solve consistently more problems in a given running time than NORELAX. The difference, however, gets smaller as the time increases. Between the two experiments SINGLE and ENERGETIC the first one is always slightly ahead of the other.

In the case an instance is not solved to optimality, the CIP approach is able to deliver feasible solutions. All four test sets contain in total 335 instances. The NORELAX experiment solved 300 instances to proven optimality. From the remaining 35 instances, this

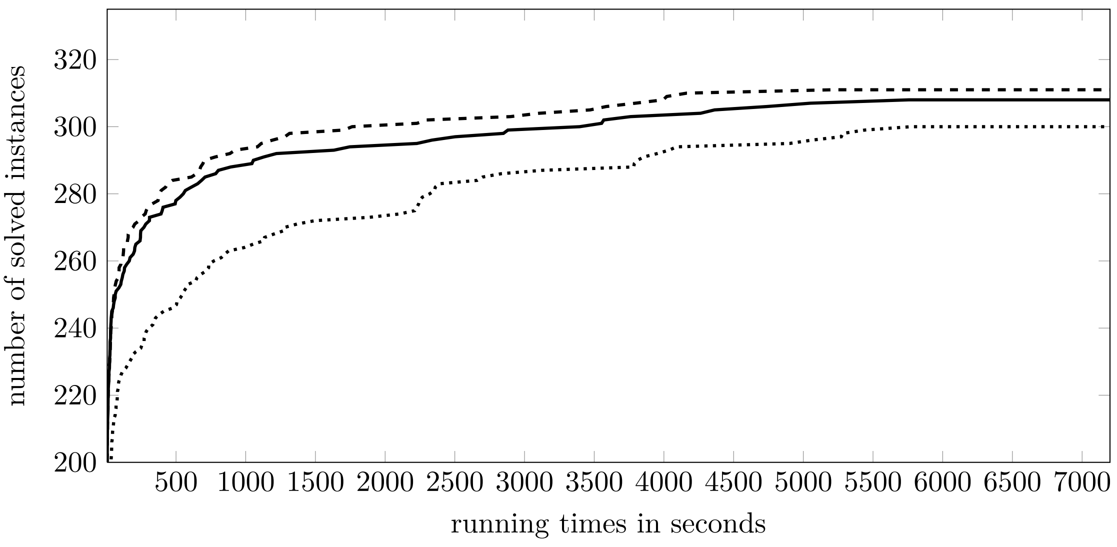


Figure 5.3: Diagram for the three CIP models (which differ in the used LP relaxation) on the 335 allocation and scheduling instances. It shows the number of instances solved within a certain time. The NORELAX model is dotted (······), the SINGLE model is dashed (- - -), and ENERGETIC model is solid (——).

setup failed on 13 instances to find a feasible solution or to proved that the problem is infeasible. Hence, for 22 instances we stop with a feasible solution available. The other two experiments SINGLE and ENERGETIC solve 311 and 308 instances, respectively. Both setups failed on 11 instances to construct a feasible solution or to prove that the corresponding instances are infeasible.

In the following section we compare the CIP approach with a LBBD and MIP method which we both discussed in the previous two sections.

5.6 Comparison

In the previous sections we presented computational studies for three approaches to tackle resource allocation and scheduling problems. We focused on a LBBD approach (see Section 5.3), a MIP approach (see Section 5.4), and a CIP approach (see Section 5.5). In this section, we compare the performance of these three approaches. Note that we use the same test sets and the same computational environment as our previous work [HB12a] allowing for a direct comparison of the results.

For each of the approaches we select one of the best setups. For the LBBD approach, this is the ENERGETIC sub-problem relaxation and the Benders cut strengthening technique. For the MIP approach we choose the EXTENDED+(5.9) model which is an extended formulation with the redundant ENERGETIC relaxation added for better propagation. We avoid any aggregations during the presolving phase as discussed in Section 5.4.3. In case of the CIP approach we use the SINGLE constraint relaxation for the cumulative constraints with optional jobs.

We compare the three approaches on the same four test set as before (see Section 5.2.1) using the same time limit of 2 hours. We use two measures to compare these 3 approaches. One is an aggregated running time to solve all instances and the other the solution quality

Table 5.20: Comparing a LBBD approach, a MIP approach, and a CIP approach for resource allocation and scheduling problems.

				LBBD			MIP			CIP		
test set	easy	hard	eval	opt	nodes	time	opt	nodes	time	opt	nodes	time
TESTSETC	40	5	150	143	15.5	26.1	129	11544.6	148.6	135	10549.4	75.4
TESTSETE	11	0	39	39	24.0	12.8	39	5072.2	56.7	35	12124.3	106.9
TESTSETDE	24	0	26	26	17.8	3.9	26	549.1	14.5	26	290.7	7.9
TESTSETDF	15	0	25	25	24.8	2.1	25	41.5	1.5	25	55.8	2.2
all	90	5	240	233	18.1	17.0	219	4615.4	75.6	221	4802.7	51.9

in cases an instance is not solved to optimality. For the used solvers and more details about the approaches we refer to corresponding sections.

5.6.1 Computational results

Table 5.20 states the summarized results for the three approaches and all four test sets (TESTSETC, TESTSETE, TESTSETDE, and TESTSETDF). For each approach we present the following information. The first column states the test set where "all" refers to all 335 instances. The columns "easy" and "hard" show the number of instances which are easy and hard. As before, an instances is seen to be easy if all three approaches solve this instance in less than one second. An instance is assigned to the hard category if all three approaches hit the time limit of 2 hours, i.e., none of them can solve this instance. The instances which belong to one of these classes are removed from the evaluation. The column "eval" states the number of instances which are part of the evaluation. The sum of "easy", "hard", and "eval" instances equals the number of instances of the corresponding test set. For the instances which belong to the evaluation we state for each approach the number of instances solved to proven optimality (column "opt") and the shifted geometric mean[7] for the number of visited search "nodes" and the overall running "time" in seconds. We applied a shift $s = 100$ for the number of search nodes and a shift $s = 10$ seconds for the overall running time. For the running time we assume a minimum running time of 0.5 seconds and instances which hit the time limit of 2 hours contribute with 7200 seconds to the shifted geometric mean of the overall running time if they are part of the evaluation set.

Overall there are 5 instances which cannot be solved by any of the used approaches. These instances belong to TESTSETC. From the remaining instances there are 90 instances categorized as easy since all three approaches solve these instances in less than one second. That leaves 240 instances for the evaluation. None of the approaches is able to solve all of these instances. The LBBD approach performs best w.r.t. proving optimality. It is able to solve 233 instances, followed by the CIP approach with 221 instances and the MIP approach which solves 219 instances. Hence, the LBBD approach solves 12 instances more than the CIP approach and 14 instances more than the MIP approach.

When taking the running time in shifted geometric mean as our measure, we get the same order. The best overall running time is achieved by the LBBD approach with 17.0 seconds, followed by the CIP approach with 51.9 seconds, and the MIP approach with 75.6 seconds. In [Hoo04, Hoo05a, CCH13] it is reported that the LBBD approach is orders of magnitudes faster than the MIP approach. Our results show a different picture because we selected an extended formulation as MIP model which differs from the previously used model. It uses

[7]The shifted geometric mean of values $t_1, \ldots, t_n$ is $\left(\prod(t_i + s)\right)^{1/n} - s$, with shift s.

Table 5.21: Comparing an LBBD approach, a MIP approach, and a CIP approach w.r.t. the 195 instances of TESTSETC. Note that several job-machine combination are omitted since all 5 instances belong to the easy category.

					LBBD			MIP			CIP		
$\|\mathcal{R}\|$	$\|\mathcal{J}\|$	easy	hard	eval	opt	nodes	time	opt	nodes	time	opt	nodes	time
2	12	4	0	1	1	6.0	0.5	1	184.0	0.5	1	22.0	1.6
	14	3	0	2	2	3.0	0.5	2	148.8	0.5	2	38.5	1.0
	16	1	0	4	4	2.5	0.8	4	911.8	2.6	4	228.4	1.7
	18	1	0	4	4	3.2	1.8	4	3511.4	18.8	4	740.6	5.4
	20	1	0	4	4	3.2	0.8	4	1854.7	8.7	4	424.7	2.6
	22	0	0	5	5	4.5	17.2	5	91281.4	949.9	5	68512.7	216.6
	24	0	0	5	5	3.2	30.8	4	19879.4	291.4	4	79575.8	361.5
	26	0	0	5	5	1.8	30.2	3	108010.8	1081.5	4	409206.5	925.0
	28	0	0	5	5	1.6	13.5	4	23795.2	248.6	4	17924.1	125.9
	30	0	0	5	4	3.8	1018.0	3	55070.4	526.3	3	34454.8	284.6
	32	0	1	4	2	1.5	553.7	4	13256.3	164.3	2	766702.9	1371.2
	34	0	1	4	3	1.2	572.9	3	43893.2	825.4	2	128161.6	750.9
	36	0	2	3	1	1.0	975.3	2	86695.6	1099.0	3	31954.3	78.8
	38	0	1	4	4	1.0	94.1	3	138622.1	2853.7	3	529645.4	732.2
3	14	2	0	3	3	10.3	0.7	3	79.2	0.7	3	139.0	1.6
	16	1	0	4	4	22.2	1.8	4	1595.8	2.1	4	881.0	3.3
	18	0	0	5	5	34.2	4.2	5	1800.6	10.8	5	2177.2	6.7
	20	0	0	5	5	11.1	1.5	5	3655.6	42.1	5	845.3	5.0
	22	0	0	5	5	19.4	1.9	5	7767.4	34.7	5	1556.5	9.0
	24	0	0	5	5	23.8	5.4	5	9940.4	102.7	5	5635.8	23.0
	26	0	0	5	5	28.1	36.8	4	127007.4	1419.8	5	79194.2	180.9
	28	0	0	5	5	10.9	24.1	3	26746.6	277.6	5	20657.4	164.3
	30	0	0	5	5	34.3	176.8	4	107766.9	1370.7	3	558026.2	1327.4
	32	0	0	5	4	57.4	603.8	3	109265.7	1830.1	3	878579.2	3622.7
4	14	2	0	3	3	14.6	1.2	3	56.7	0.5	3	123.9	2.0
	16	0	0	5	5	7.2	0.7	5	68.1	0.7	5	158.9	2.0
	18	0	0	5	5	19.3	1.8	5	410.4	1.6	5	388.5	3.1
	20	0	0	5	5	14.3	1.5	5	1604.1	6.8	5	602.2	4.4
	22	0	0	5	5	16.2	1.9	5	1620.1	13.1	5	954.7	6.7
	24	0	0	5	5	42.9	8.7	4	27051.2	253.0	5	11252.1	47.1
	26	0	0	5	5	43.1	12.1	4	109549.3	845.2	5	16801.0	47.0
	28	0	0	5	5	14.4	9.2	5	21687.0	284.1	5	6286.7	36.1
	30	0	0	5	5	36.8	49.7	3	121184.5	1611.2	5	53458.5	264.6
	32	0	0	5	5	19.0	85.9	3	173367.9	2492.0	4	362364.1	1152.4
TESTSETC		40	5	150	143	15.5	26.1	129	11544.6	148.6	135	10549.4	75.4

the same improved relaxation as we use in the LBBD approach. This model is not able to produce the same performance as the LBBD approach on the selected instances but is clearly not orders of magnitudes slower. We observe a factor of less than 4.5 comparing the LBBD and MIP approach using the running time as measure.

In the following we analyze the performance of the different approaches for the individual test sets in more detail, including the solution quality in cases an instance is not solved to proven optimality.

Test set testsetc. Table 5.21 states the results for TESTSETC. This test set has 195 instances. A characteristic of these instances is that all jobs have the same release and due date.

In total there are 5 instances categorized as hard since none of the three approaches can solve these instances within the time limit of 2 hours. These instances belong to the class of instances which have 2 machines. There are 40 instances in the easy category since all approaches solve these instances in less than one second. That leaves 150 instances for the evaluation.

The best performance w.r.t. running time and number of solved instances is achieved by the LBBD approach. It solves 143 of the evaluation instances and needs an aggregated time of 26.1 seconds. The MIP approach solves 129 instances (14 instances less than the

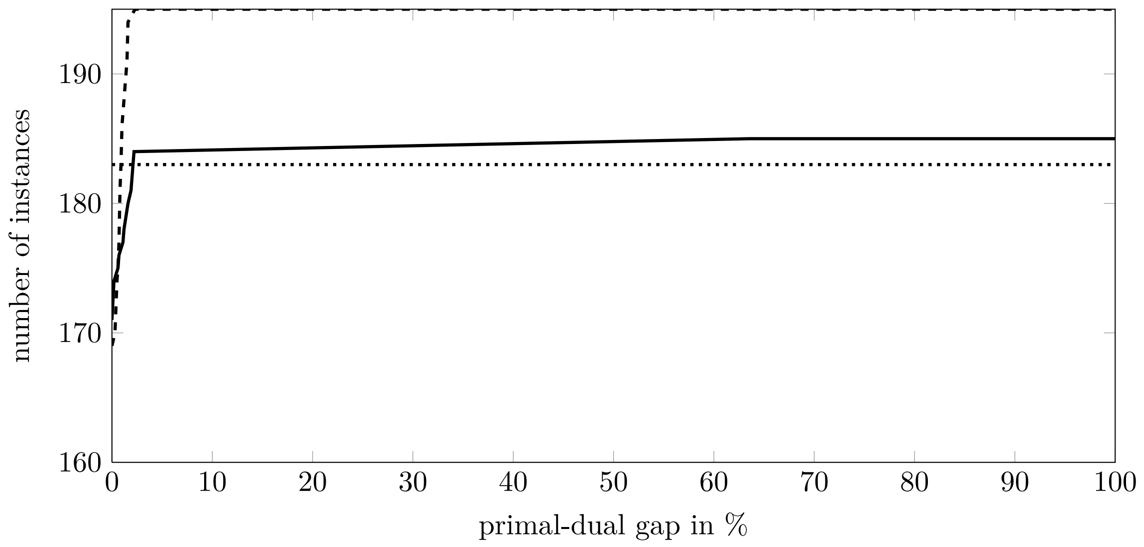


Figure 5.4: Diagram for the three approaches: LBBD, MIP, and CIP on the 195 allocation and scheduling instances of TESTSETC. It displays the number of instances for which a solution with a given optimality gap or better was found. The LBBD approach is dotted (······), the MIP approach is dashed (- - -), and CIP approach is solid (——).

LBBD approach) and the CIP is capable to solve 135 instances (8 instances less than the LBBD approach). Comparing the time measure, the MIP and CIP approach are almost a factor 6 and 3 slower than the LBBD approach, respectively.

For the MIP and CIP approach it can be observed that in general the running time and the required number of search nodes clearly increase if more jobs need to be scheduled. For the LBBD approach the increase in the running time can also be seen but in a much smaller manner. Interestingly, there are job-machine combinations with 2 machines where MIP or CIP approach dominates the other approaches.

Figure 5.4 visualizes the primal-dual gap for the three approaches on all 195 instances. The primal-dual gap is defined as the primal bound minus the dual bound divided by the primal bound. The figure displays the number of instances for which a solution with a given primal-dual gap or better was found. The LBBD approach is dotted (······), the MIP approach is dashed (- - -), and CIP approach is solid (——). Note that the LBBD approach is designed to only find optimal solutions. In cases where an instance is not solved to proven optimality the approach does not provide any feasible solution. In contrast the MIP and CIP approach delivers feasible suboptimal solutions. The figure shows that even in those cases where the LBBD approach is the only one that solves an instance, the other two approaches are able to provide high quality solutions. The MIP approach finds a feasible solution or proves infeasibility for all instances. The largest primal-dual gap is 2.3%. The CIP approach fails for 10 instances to find a feasible solution or prove infeasibility. For one instance the primal-dual gap is 63.6%. For the remaining 174 instances the primal-dual gap is at most 2.2%.

Overall, the LBBD approach gives the best performance considering the running time to proven optimality. The other approaches also show a reasonable performance and are able to solve instances which are not solved by the LBBD approach. Taking the primal-dual gap into account, the MIP approach dominates the other approaches. It finds high quality solutions for all instances or proves infeasibility. The LBBD and CIP approach fail to find a feasible solution for 12 and 10 instances, respectively. We conclude that the MIP approach

Table 5.22: Comparing an LBBD approach, a MIP approach, and a CIP approach w.r.t. the 50 instances of TESTSETE. Note that the job-machine combination with 2 machines and 10 jobs is omitted since all 5 instances belong to the easy category.

					LBBD			MIP			CIP		
$\lvert\mathcal{R}\rvert$	$\lvert\mathcal{J}\rvert$	easy	hard	eval	opt	nodes	time	opt	nodes	time	opt	nodes	time
2	12	4	0	1	1	22.0	1.0	1	274.0	0.9	1	112.0	1.2
3	15	2	0	3	3	10.8	1.0	3	48.8	0.7	3	279.4	2.0
4	20	0	0	5	5	20.3	2.8	5	1199.9	4.9	5	692.0	6.5
5	25	0	0	5	5	9.1	0.9	5	1388.2	4.9	5	273.6	3.6
6	30	0	0	5	5	12.1	1.7	5	1603.5	10.0	5	2068.1	11.7
7	35	0	0	5	5	28.9	7.8	5	9346.5	80.6	5	28192.3	100.4
8	40	0	0	5	5	41.1	46.0	5	18124.0	169.2	4	191942.9	821.0
9	45	0	0	5	5	46.5	103.6	5	55259.3	699.6	3	816834.1	3692.5
10	50	0	0	5	5	23.1	21.1	5	44726.2	484.1	4	258827.3	1112.8
TESTSETE		11	0	39	39	24.0	12.8	39	5072.2	56.7	35	12124.3	106.9

Table 5.23: Comparing an LBBD approach, a MIP approach, and a CIP approach w.r.t. the 50 instances of TESTSETDE. Note that the job-machine combination with 3 machines and 10 and 12 jobs is omitted since all 5 instances belong to the easy category.

					LBBD			MIP			CIP		
$\lvert\mathcal{R}\rvert$	$\lvert\mathcal{J}\rvert$	easy	hard	eval	opt	nodes	time	opt	nodes	time	opt	nodes	time
3	14	4	0	1	1	13.0	0.7	1	63.0	0.6	1	28.0	1.1
3	16	4	0	1	1	9.0	0.6	1	1.0	0.5	1	47.0	1.4
3	18	3	0	2	2	5.5	0.6	2	3.0	0.6	2	105.8	2.7
3	20	1	0	4	4	9.0	0.8	4	60.6	1.4	4	88.7	2.0
3	22	2	0	3	3	12.4	1.2	3	87.5	1.4	3	112.1	3.3
3	24	0	0	5	5	6.7	1.9	5	1033.8	26.4	5	89.2	4.8
3	26	0	0	5	5	40.2	11.9	5	1033.2	20.3	5	1085.1	18.7
3	28	0	0	5	5	28.3	7.8	5	5376.8	66.5	5	1249.8	21.0
TESTSETDE		24	0	26	26	17.8	3.9	26	549.1	14.5	26	290.7	7.9

Table 5.24: Comparing an LBBD approach, a MIP approach, and a CIP approach w.r.t. the 40 instances of TESTSETDF. Note that the job-machine combination with 3 machines and 16 jobs is omitted since all 5 instances belong to the easy category.

					LBBD			MIP			CIP		
$\lvert\mathcal{R}\rvert$	$\lvert\mathcal{J}\rvert$	easy	hard	eval	opt	nodes	time	opt	nodes	time	opt	nodes	time
3	14	4	0	1	1	27.0	1.1	1	1.0	0.5	1	6.0	0.5
3	18	2	0	3	3	24.3	1.6	3	1.7	0.7	3	52.1	1.8
3	20	2	0	3	3	25.3	1.6	3	4.2	0.8	3	28.0	1.0
3	22	0	0	5	5	20.4	1.9	5	38.8	1.1	5	53.5	2.0
3	24	1	0	4	4	54.0	4.8	4	167.1	1.6	4	112.7	4.9
3	26	1	0	4	4	17.6	1.9	4	36.2	1.5	4	54.3	1.6
3	28	0	0	5	5	14.4	1.5	5	40.4	3.1	5	53.2	2.2
TESTSETDF		15	0	25	25	24.8	2.1	25	41.5	1.5	25	55.8	2.2

is the most robust approach for this test set.

Test set testsete. This set contains 50 instances. As for the instances of TESTSETC all jobs have the same release and due date. In this test set the number of machines and jobs increase up to 10 and 50, respectively. Table 5.22 states the detailed results.

None of the instances is categorized as hard whereas 11 instances are easy since all three approaches can solve them in less than a second. That leaves 39 instances for the evaluation. Since all instances are solved by all three approach, we do not need to analyze the solution quality for instances which are not solved to proven optimality.

The LBBD approach and the MIP approach are able to solve all these instances. The CIP approach fails on 4 instances. As we analyzed in the previous section, these fails results from a poor performance of the primal heuristics (see Section 5.5.3).

The CIP approach gives the worst performance w.r.t. the running time. This approach needs 106.9 seconds in shifted geometric mean. The MIP approach is a factor 2 faster with an aggregated running time of 56.7 seconds. The best performance w.r.t. running time is achieved by the LBBD approach which needs 12.8 seconds. That is a factor 5 faster than the MIP approach and a factor 9 faster than the CIP approach.

For all approaches it can be observed that the running time in general increases if the problems get larger. Interestingly, the largest class of instances (10 machines and 50 jobs) is solved by all approaches better than the second largest problem class.

Test set testsetde. In contrast, the two previous test sets, the instances of TESTSETDE have the same release date but differ in their due dates. In total this test set has 50 instances. Table 5.23 presents the results for these instances.

All three approaches are able to solve all 50 instances. From these instances 24 instances are categorized as easy since all approaches solves them in less than a second. That leaves 26 instances for the evaluation.

The best running time w.r.t. the shifted geometric mean is achieved by the LBBD approach with 3.9 seconds. A factor 2 slower is the CIP approach with 7.9 seconds. The MIP approach needs 14.5 seconds in shifted geometric mean which is almost a factor 4 slower than the LBBD approach. Overall, all three approach give reasonable results.

Test set testsetdf. The last test set TESTSETDF contains 40 instances. These instances have different release and due dates. Table 5.24 summarizes the results for them.

All instances can be solved by all three approaches. In total there are 15 instances categorized as easy since these instances are solved by each approach in less than one second. For the remaining 25 instances all approach show a similar performance. The MIP approach needs 1.5 second in shifted geometric mean which is slightly faster than the other two approaches which take just a little more than 2 seconds.

For this test set we cannot observe that the running time increases for larger instances.

5.6.2 Conclusions

The results of the experiments presented in this chapter show that both CIP and MIP should be considered to be the state-of-art models, along with LBBD, for the tested resource allocation and scheduling problems. To come to these conclusions, we used two primary measures of model performance: the number of problem instances solved to proven optimality and the proven quality of solutions found, given that not all instances were solved to optimality. CIP comes second to LBBD by the former measure and to MIP by the latter. Figure 5.5 shows the evolution of the number of problems solved to optimality over time. It can be observed that LBBD dominates CIP and CIP dominates MIP w.r.t. this measure. TESTSETC is the only test set where each of the approaches fails to solve all instances. Figure 5.4 visualizes the solution quality for all instances of this test set. Considering this measure, MIP dominates CIP and CIP dominates LBBD.

Depending on the importance placed on these measures any of the three algorithms could be declared the “winner”. For practical purposes, we believe that the importance of proven solution quality should not be under-estimated: in an industrial context it is typically better to consistently produce proven good solutions than to often find optimal solutions but sometimes fail to find any feasible solution at all.

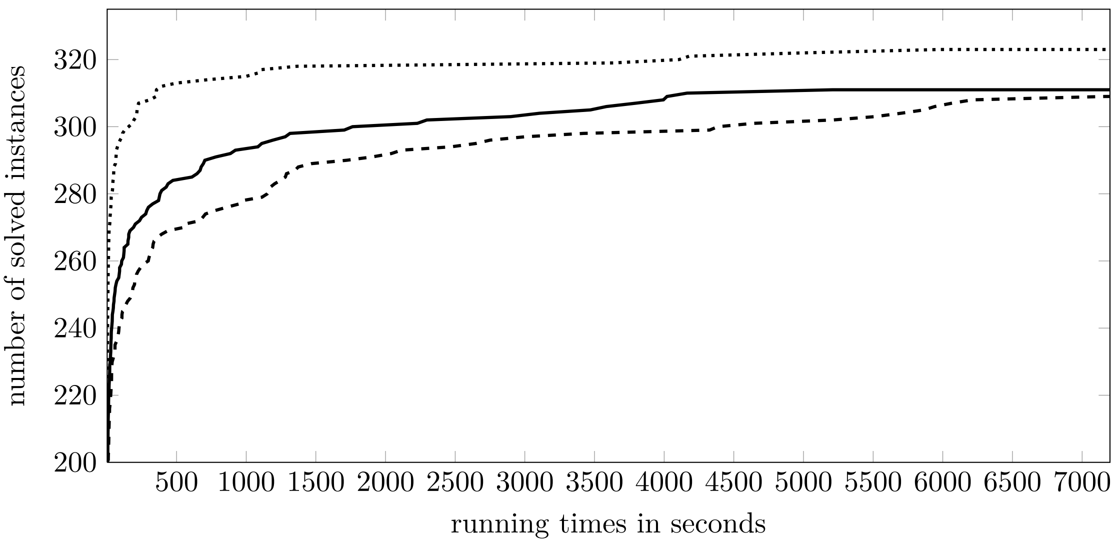


Figure 5.5: Diagram for the three approaches: LBBD, MIP, and CIP on the 335 allocation and scheduling instances. It shows the number of instances solved within a certain time. The LBBD approach is dotted (······), the MIP approach is dashed (- - -), and CIP approacheis solid (——).

An examination of the sub-problems in two-resource instances of test set TESTSETC that LBBD and CIP fail to solve reveals that most of the cumulative constraint/sub-problems which have to be proven to be feasible or infeasible have a very small slack and the jobs have wide and often identical time windows. Slack is the difference between the rectangular area available on the resource (length of effective horizon time by capacity) and the sum of the areas (processing time by resource requirement) of the jobs. Alternatively, slack can be understood to be the tightness of the single relaxation (Equation (2.18)). The small slack results from the fact that one resource is consistently less costly than the others and so it appears promising to assign as many jobs as possible within the limits of the interval relaxation. This slack is not equivalent to the flexibility of a job w.r.t. a resource which we defined in Section 5.3.2 and used to create an ordering for a set of jobs..

All approaches suffer from not being able to handle small slack and wide time window problems efficiently on cumulative resources. This is the underlying reason for the occasional failure of LBBD. It gets stuck at such sub-problems and fails completely to find a feasible solution. All other approaches have the same issue of not being able to solve these implicit sub-problems, but are able to provide high quality primal solutions. To overcome this issue, stronger cumulative inference techniques [BCP08] may be worth consideration.

As we are comparing the CIP approach against a start-of-the-art LBBD implementation, we should also compare it with a state-of-the-art commercial MIP solver when solving MIP models. It has been standard for the past 10 years for commercial MIP solvers to use multiple cores. If we run IBM ILOG CPLEX with its default settings (using all available cores, eight in our case) on all instances we can solve 317 instances to proven optimality with a shifted geometric mean of 44.0 seconds (see Table 5.14). The fact that this performance w.r.t. number of solved instances is only marginally better than what we observe for CIP and similar to LBBD results, strengthens the claim to be a state-of-the-art approach.

Finally, if the best of the three models is chosen individually for each instance (resulting in the *virtual best solver*), 330 instances can be solved demonstrating that none of the models is dominant.

Summary and outlook

In this dissertation, we focused on primal and dual inference techniques for the cumulative constraint which can be used to describe a relationship between a renewable resource, e.g., a machine, and non-preemptive jobs which require a certain amount of the resource during their execution. We restricted ourselves to a setup where the available resource capacity is constant over time and where each job has a fixed processing time and resource demand (Section 1.4.1). We allowed, however, that jobs can be optional for a resource requiring that a decision has to be made whether a job is processed or not (Section 1.4.4).

In the following we summarize our contributions and give a outlook for future work.

Contributions

Achterberg [Ach07b] introduced the concept of variable locks for a constraint integer program (CIP) to define dual reductions based on global information for mixed-integer programs (MIP). Essentially, a variable lock represents information about the relationship between a variable and a set of constraints. Building on the existing idea of variable locks, we formally defined and justified the use of dual information for constraint optimization problems (COP) in Section 1.3.2.

In Chapter 2 we discussed two relaxations for the cumulative constraint: a conflict relaxation and a linear relaxation. For our systematic approach, we formally defined energy-based propagation algorithms (Definition 2.1), a class of propagation algorithms for cumulative constraints. These algorithms use specific energy-based arguments to infer bound changes or to detect infeasibilities.

For the conflict relaxation, which is a set of conflict constraints retrieved from the analysis of infeasible sub-problems, we developed generic explanations (Definition 2.6) for the inferences made by energy-based propagation algorithms (Theorem 2.8 and Theorem 2.9). These general explanations rely on the observation that all these algorithms use a lower bound on the energy consumption of a fixed job j within a fixed interval and that the volume formulas used for inference are the same (see Section 2.1). For particular algorithms these general explanations are identical to previously published explanations. The added value lies in the generalization to a whole class of propagation algorithms.

For the linear relaxation of cumulative constraints with optional jobs, we used the same observation as for the conflict relaxation and showed how to construct a linear relaxation from any propagation algorithm belonging to the class of energy-based propagation algorithms, see Section 2.3.2. This general concept captures known linear relaxations for this structure. In addition, we presented a new linear relaxation which is a corollary of our concept using the energetic reasoning propagation algorithm (Inequalities 2.23).

Chapter 3 is dedicated to generic presolving techniques for the cumulative constraint. In Section 3.3 we adapted the notions of variable locks and variable bounds to the cumulative constraint. In addition, we generalize known problem specific presolving techniques for the cumulative constraint, thus, making them available within a general purpose solver through the use of globally available structures, i.e., the variable bound graph. In Section 3.4, we applied the concept of dual reductions described in Section 1.7 for generic COPs and presented several dual reductions for the cumulative constraint. We developed two types of dual reductions: one which analyzes the setup w.r.t. the effective horizon (Definition 1.19), a concept introduced in this dissertation, and the other which infers schedule-or-postpone situations. The effective horizon is also used to decompose cumulative constraints (Lemma 3.7). These reductions are applicable to a single cumulative constraint, but are also generalized to a set of cumulative constraints. Section 3.5 stated presolving methods for the cumulative constraint with optional jobs. These methods rely on presolving techniques for the cumulative constraint without optional jobs.

Chapter 4 contributed an extensive computational study of resource-constrained project scheduling problems (Section 4.1) using the constraint integer programming solver SCIP. We focused on three topics: (i) we evaluated the importance of the different propagation algorithms available for cumulative constraints for resource-constrained project scheduling problems; (ii) we analyzed the impact of a presolving phase; (iii) we presented results which compare SCIP against a state-of-the-art solver for resource-constrained project scheduling problems.

We observed that the edge-finding propagation algorithm in its implementation in SCIP is not important at all for resource-constrained project scheduling problems. Disabling this algorithm leads to a better performance for both considered sets (Section 4.2.1). This is surprising since in the literature [Vil11, SFS13] it is reported that this algorithm is crucial to solve some of the instances of the Psplib.

We evaluated the impact of different dual reductions for resource-constrained project scheduling problems (Section 4.4), utilizing variable locks and the effective horizon. Our experiments showed that these reductions often occur for resource-constrained project scheduling problems with standard precedence constraints and less frequently for resource-constrained project scheduling problems with generalized precedence constraints. The results further showed that our presolving techniques are able to find additional variable fixings during the presolving phase.

In Section 4.5 we compared the capability of SCIP, as a cumulative scheduling solver to a problem specific state-of-the-art implementation. The results clearly showed that SCIP can be seen as a state-of-the-art solver for resource-constrained project scheduling problems in its standard version as well as with generalized precedence constraints.

Resource allocation and scheduling problems were considered in Chapter 5. We considered three approaches to tackle this problem class: a logic-based Benders decomposition, a mixed-integer programming approach, and a constraint integer programming model.

In Section 5.3, we used the linear relaxation for the cumulative constraint with optional jobs, developed in Section 2.3, for an improved sub-problem relaxation within an LBBD approach. Empirical results showed that the improved sub-problem relaxation is the best choice. In Section 5.4, we discussed mixed-integer programming formulations for the resource allocation and scheduling problem under investigation. We used a known model [HO03] for this problem and analyzed its behavior for modern MIP solvers. As a result we suggested an extended model which is superior to previously known models. In Section 5.5, we presented a constraint integer programming model which uses the cumulative constraint with optional jobs.

The results of the experiments presented in this chapter showed that both CIP and MIP should be considered state-of-art models, along with LBBD, for the tested resource allocation and scheduling problems.

Future work

The contribution of this thesis comprises three topics: dual reductions for constraint programs, the analysis of infeasible sub-problems with cumulative constraints, and using constraint integer programming to solve scheduling problems efficiently. For each of these topics, we see numerous possibilities for further research.

Dual reductions

Dual reductions are a form of problem reformulation that removes problem components (e.g. variables and/or values) and perhaps feasible and optimal solutions while guaranteeing that at least one optimal solution remains in the transformed search space (provided the problem is feasible). While there has been work in constraint programming (CP) on techniques that can be understood to be dual reductions, notably in the form of symmetry breaking, neither dual reductions nor their common implementation in presolving for MIP solvers has received much attention in the CP community. We believe that both presolving and dual reductions are promising general concepts to be implemented in a generic CP solver.

In this dissertation, we formalized variable locks by defining constraints that are monotonously decreasing or increasing in a variable (Definition 1.11), as a means of collecting information for constraint programs.

One essential condition for variable locks is that the variable domain is totally ordered. For problems without such an ordering, one could artificially introduce such an order and apply the concepts of variable locks. Consider for example the all-different constraint [Lau78, vH01]. For a given set of variables, it enforces that each variable takes a different value and, as a consequence, the domain of the variables does not need to be totally ordered. Introducing an order, however, could lead to dual fixings, as shown in the following example.

Example 5.3. Consider the variables x_1 with domain $\{1, 3, 5\}$ and x_2 with a domain of $\{2, 3, 4\}$. Using the natural ordering of the numbers, an all-different constraint over these two variables has to lock both variables in both directions. Changing the domain orders to $\{3, 1, 5\}$ for x_1 and to $\{2, 4, 3\}$ for x_2, an all-different constraint over these two variables does not need to up-lock variable x_1 and does not need to down up-lock variable x_2.

The question arising from this observation is, which order on the domain values is the best one w.r.t. dual reasoning.

The combination of variable locks and global constraints in CP is a novel aspect of this work. Our development and analysis of valid dual reductions relies directly on the semantics of the cumulative constraint whereas dual reasoning in MIP is based on the comparatively limited structure embodied by a linear constraint. We believe therefore that, just as standard constraint inference relies on the meaning of a global constraint, there is a rich vein of reasoning that can be done by deriving and combining dual information based on the semantics of a global constraint.

Variable locks are one way for a constraint-based system to collect dual information. We showed how such information can be used for the cumulative constraint. A next step is to analyze other global constraints and develop other concepts that provide similar information for a constraint-based system.

Analysis of infeasible sub-problems

Learning from infeasible sub-problems is a powerful tool for any solver that is based on a tree search. The analysis of infeasible sub-problems generates conflict constraints which need to be fulfilled by any feasible solution and form a relaxation of the original problem. This relaxation provides statistics about the distribution of variables in conflict constraints which can be used to drive the search (Section 2.2.1). To do a conflict analysis as described in Section 2.2, it is crucial, that for each bound change, an explanation (Definition 2.6) is present. These explanations can be represented as an implication graph containing the information which sets of bound changes force the problem to be infeasible. For a clause as it appears in a SAT problem, there exists a unique explanation for each bound change derived from unit propagation. For a (general) linear constraint or the cumulative constraint considered in this dissertation, an explanation does not need to be unique: there are many possible implication graphs. This observation raises the question which explanation should be used:

- ▷ Should an explanation be small w.r.t. the number of bound changes?
- ▷ If there are several explanations of the same size, which is the better one?
- ▷ Should the construction of an explanation consider the structure of the current search tree since each bound change is associated with a search node?
- ▷ Should an explanation be chosen which results in a minimum extension of the implication graph?

Explanations as defined in this dissertation (Definition 2.6) are limited to a set of bound changes where bound changes can be seen as the primitives of the solver for domain propagation. If an overload is detected by a cumulative constraint within a time interval, it often is clear that symmetric overloads exist by shifting all jobs to the left or to the right w.r.t. their possible start time. Hence, such an overload can be seen as a representative of a set of overloads.

Example 5.4. Consider three jobs each with a unit demand and a processing time of 5 units. Job 1 has a release date of 0 and a due date of 10, job 2 is released at time point 5 and must be finished by time point 15, and job 3 is available at time point 0 and has a due date of 15. Assume that these three jobs need be processed by a resource with a capacity of 2. At most two jobs can be processed in parallel.

An overload w.r.t. the time interval $[7, 8)$ is given if the earliest start times of job 1 and 3 are shifted to 3, and the latest start time of job two and three is moved to 7. If we denote the start time variables for the three jobs with S_1, S_2, and S_3, the overload can be explained with the following bound changes:

$$[\![S_1 \geq 3]\!] \cap [\![S_3 \geq 3]\!] \cap [\![S_2 \leq 7]\!] \cap [\![S_3 \leq 7]\!].$$

From the structure of the cumulative constraint and the given (global) bounds for the start time variables, this overload can seen as representative of a set of overloads. For each $t \in \{-3, -2, -1, 0, 1, 2, 3\}$ the following bound changes lead to an overload for the cumulative constraint in the time interval $[7 + t, 8 + t)$:

$$[\![S_1 \geq 3 + t]\!] \cap [\![S_3 \geq 3 + t]\!] \cap [\![S_2 \leq 7 + t]\!] \cap [\![S_3 \leq 7 + t]\!].$$

The question is whether the idea of generating a set of alternative explanations can be realized efficiently and whether it helps to improve the performance of a solver.

CIP for scheduling

There are a number of areas of future work both on extending the LBBD, MIP, and CIP approaches to related scheduling problems and in developing the technology of CIP for scheduling.

We have demonstrated through the integration of the `optcumulative` constraint that global constraint-based presolving, inferences, and relaxations can lead to state-of-the-art performance. We believe that the further integration of global constraint reasoning into a CIP framework for scheduling and other optimization problems is a promising field for future research.

In [Hoo11], Hooker presented LBBD models for extensions of the problem studied here with a number of different optimization functions. For such problems, LBBD is able to produce feasible sub-optimal solutions during the solving process without necessarily finding an optimal solution. Therefore, one of the main advantages, finding feasible sub-optimal solution during the solving process, of the MIP and CIP techniques compared to LBBD does not appear. It will be valuable to understand how adaptations of the MIP and CIP models presented here perform for these extended problems. Another important class of scheduling problems include temporal constraints among jobs on different resources. Such constraints destroy the independent sub-problem structure that LBBD and, to a lesser extent, the other models exploit. Exact techniques struggle on such problems including flexible job shop scheduling [FSMJ07].

Our results for resource-constrained project scheduling problems and resource allocation and scheduling problems showed that all models are unable to find optimal solutions as the number of jobs scales. With even more jobs, the only achievable performance measure will be the quality of feasible solutions that are found. We expect LBBD to perform poorly given that it does not provide sub-optimal solutions for resource allocation and scheduling problems. However, as the problem size increases the time-indexed formulation on the MIP model will also fail due to the sheer model size. CIP and the pure CP model [HB12a] would appear to be the only exact techniques likely to continue to deliver feasible solutions. Confirming this conjecture, as well as comparing the model performance to incomplete techniques (i.e., heuristic and metaheuristics) is therefore another area for future work.

Bibliography

[AB93] Abderrahmane Aggoun and Nicolas Beldiceanu. Extending chip in order to solve complex scheduling and placement problems. *Mathematical and Computer Modelling*, 17(7):57–73, 1993.

[AB09] Tobias Achterberg and Timo Berthold. Hybrid branching. In Willem Jan van Hoeve and John N. Hooker, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 6th International Conference, CPAIOR 2009*, volume 5547 of *Lecture Notes in Computer Science*, pages 309–311. Springer, May 2009.

[ABW07] Tobias Achterberg, Raik Brinkmann, and Markus Wedler. Property checking with constraint integer programming. ZIB-Report 07-37, Zuse Institute Berlin, 2007.

[Ach04] Tobias Achterberg. SCIP – a framework to integrate Constraint and Mixed Integer Programming. ZIB-Report 04-19, Zuse Institute Berlin, 2004.

[Ach07a] Tobias Achterberg. Conflict analysis in mixed integer programming. *Discrete Optimization*, 4(1):4–20, 2007.

[Ach07b] Tobias Achterberg. *Constraint Integer Programming.* PhD thesis, Technische Universität Berlin, 2007.

[Ach09] Tobias Achterberg. SCIP: Solving Constraint Integer Programs. *Mathematical Programming Computation*, 1(1):1–41, 2009.

[ADN08] Christian Artigues, Sophie Demassey, and Emmanuel Néron, editors. *Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications.* iSTE, 2008.

[AMR03] Christian Artigues, Philippe Michelon, and Stéphane Reusser. Insertion techniques for static and dynamic resource constrained project scheduling. *European Journal of Operational Research*, 149:249–267, 2003.

[AR10] Tobias Achterberg and Christian Raack. The mcf-separator: detecting and exploiting multi-commodity flow structures in MIPs. *Math. Program. Comput.*, 2(2):125–165, 2010.

[AW13] Tobias Achterberg and Roland Wunderling. Mixed integer programming: Analyzing 12 years of progress. In Michael Jünger and Gerhard Reinelt, editors, *Facets of Combinatorial Optimization*, pages 449–481. Springer Berlin Heidelberg, 2013.

[Bal05] Egon Balas. Projection, lifting and extended formulation in integer and combinatorial optimization. *Annals of Operations Research*, 140(1):125–161, 2005.

[BCDP07] Nicolas Beldiceanu, Mats Carlsson, Sophie Demassey, and Thierry Petit. Global constraint catalogue: Past, present and future. *Constraints*, 12(1):21–62, 2007.

[BCM08] Lucas Bordeaux, Marco Cadoli, and Toni Mancini. A unifying framework for structural properties of csps: definitions, complexity, tractabilit. *J. Artif. Int. Res.*, 32(1):607–629, June 2008.

[BCP08] Nicolas Beldiceanu, Mats Carlsson, and Emmanuel Poder. New filtering for the cumulative constraint in the context of non-overlapping rectangles. In *Proceedings of the 5th international conference on Integration of AI and OR techniques in constraint programming for combinatorial optimization problems*, CPAIOR'08, pages 21–35, 2008.

[BDM$^+$99] Peter Brucker, Andreas Drexl, Rolf Möhring, Klaus Neumann, and Erwin Pesch. Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, 112(1):3–41, 1999.

[Bec99] J. Christopher Beck. *Texture measurements as a basis for heuristic commitment techniques in constraintdirected scheduling.* PhD thesis, University of Toronto, 1999.

[Bec10] J. Christopher Beck. Checking-up on branch-and-check. In David Cohen, editor, *Principles and Practice of Constraint Programming – CP 2010*, volume 6308 of *LNCS*, pages 84–98. Springer, 2010.

[Ben62] Jacques F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962.

[Ben05] Jacques F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Computational Management Science*, 2(1):3–19, 2005.

[Ber06] Timo Berthold. Primal Heuristics for Mixed Integer Programs. Master's thesis, Technische Universität Berlin, 2006.

[Ber14] Timo Berthold. *Heuristic algorithms in global MINLP solvers.* PhD thesis, Technische Universität Berlin, 2014.

[BF00a] J. Christopher Beck and Mark. S. Fox. Constraint directed techniques for scheduling with alternative activities. *Artificial Intelligence*, 121:211–250, 2000.

[BF00b] J. Christopher Beck and Mark. S. Fox. Dynamic problem structure analysis as a basis for constraint-directed scheduling heuristics. *Artificial Intelligence*, 117(1):31–81, 2000.

[BFG$^+$00] Robert E. Bixby, Mary Fenelon, Zonghao Gu, Ed Rothberg, and Roland Wunderling. MIP: Theory and practice – closing the gap. In Michael J. D. Powell and Stefan Scholtes, editors, *Systems Modelling and Optimization: Methods, Theory, and Applications*, pages 19–49. Kluwer Academic Publisher, 2000.

[BGG$^+$71] Michel Bénichou, Jean-Michel Gauthier, Paul Girodet, Gerard Hentges, Gerard Ribière, and O. Vincent. Experiments in mixed-integer linear programming. *Mathematical Programming*, 1(1):76–94, 1971.

[BHL$^+$10] Timo Berthold, Stefan Heinz, Marco E. Lübbecke, Rolf H. Möhring, and Jens Schulz. A constraint integer programming approach for resource-constrained project scheduling. In Andrea Lodi, Michela Milano, and Paolo Toth, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 6140 of *Lecture Notes in Computer Science*, pages 313–317. Springer, 2010.

[BHS11] Timo Berthold, Stefan Heinz, and Jens Schulz. An approximative criterion for the potential of energetic reasoning. In Alberto Marchetti-Spaccamela and Michael Segal, editors, *Theory and Practice of Algorithms in (Computer) Systems*, volume 6595 of *Lecture Notes in Computer Science*, pages 229–239. Springer, 2011.

[BHvMW09] Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, February 2009.

[Blo70] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, July 1970.

[BP00] Philippe Baptiste and Claude Le Pape. Constraint propagation and decomposition techniques for highly disjunctive and highly cumulative project scheduling problems. *Constraints*, 5(1/2):119–139, 2000.

[BP06] A. Bockmayr and N. Pisaruk. Detecting infeasibility and generating cuts for mixed integer programming using constraint programming. *Computers & Operations Research*, 33:2777–2786, 2006.

[BPN01] Philippe Baptiste, Claude Le Pape, and Wim Nuijten. *Constraint-based Scheduling*. Kluwer Academic Publishers, 2001.

[Bru01] Peter Brucker. *Scheduling Algorithms*. Springer-Verlag New York, Inc., 3rd edition, 2001.

[BT01] James E. Borrett and Edward P. K. Tsang. A context for constraint satisfaction problem formulation selection. *Constraints*, 6(4):299–327, 2001.

[Car82] Jacques Carlier. One machine problem. *European Journal of Operational Research*, 11:42–47, 1982.

[CCH13] André Ciré, Elvin Coban, and John N. Hooker. Mixed integer programming vs. logic-based benders decomposition for planning and scheduling. In Carla Gomes and Meinolf Sellmann, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 7874 of *Lecture Notes in Computer Science*, pages 325–331. Springer, 2013.

[CCZ13] Michele Conforti, Gérard Cornuéjols, and Giacomo Zambelli. Extended formulations in combinatorial optimization. *Annals of Operations Research*, 204(1):97–143, 2013.

[Dak65] Robert J. Dakin. A tree-search algorithm for mixed integer programming problems. *The Computer Journal*, 8(3):250–255, 1965.

[Dan51] George B. Dantzig. Maximization of a linear function of variables subject to linear inequalities. In T.C. Koopmans, editor, *Activity Analysis of Production and Allocation*, pages 339–347. John Wiley & Sons, New York, 1951.

[DB97] R. Debruyne and C. Bessière. Some practicable filtering techniques for the constraint satisfaction problem. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI97)*, pages 412–417, 1997.

[DDK12] Ulrich Dorndorf, Stefan Droste, and Thorsten Koch. *Using Zimpl for Modeling Production Planning Problems*, pages 145–158. Springer, Berlin, Heidelberg, 2012.

[Dec03] Rina Dechter. *Constraint processing.* Elsevier Morgan Kaufmann, 2003.

[FS09] Thibaut Feydy and Peter J. Stuckey. Lazy clause generation reengineered. In *Proceedings of the 15th international conference on Principles and practice of constraint programming*, CP'09, pages 352–366. Springer-Verlag, 2009.

[FSMJ07] P. Fattahi, M. Saidi Mehrabad, and F. Jolai. Mathematical modeling and heuristic approaches to flexible job shop scheduling problems. *Journal of Intelligent Manufacturing*, 18(3):331–342, 2007.

[GBHW15] Gerald Gamrath, Timo Berthold, Stefan Heinz, and Michael Winkler. Structure-based primal heuristics for mixed integer programming. In *Optimization in the Real World*, volume 13, pages 37–53. 2015.

[Geo72] A. M. Geoffrion. Generalized Benders decomposition. *Journal of Optimization Theory and Applications*, 10(4):237–260, 1972.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-Completeness.* W.H. Freeman and Company, New York, 1979.

[GN07] Eugene Goldberg and Yakov Novikov. BerkMin: A fast and robust sat-solver. *Discrete Applied Mathematics*, 155(12):1549–1561, 2007.

[GNS96] Zonghao Gu, George L. Nemhauser, and Martin W. P. Savelsbergh. Lifted flow cover inequalities for mixed 0-1 integer programs. *Mathematical Programming*, 85:439–467, 1996.

[HB11] Stefan Heinz and J. Christopher Beck. Solving resource allocation/scheduling problems with constraint integer programming. In Miguel A. Salido, Roman Barták, and Nicola Policella, editors, *Proceedings of the Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems (COPLAS 2011)*, pages 23–30, 2011.

[HB12a] Stefan Heinz and J. Christopher Beck. Reconsidering mixed integer programming and MIP-based hybrids for scheduling. In Nicolas Beldiceanu, Narendra Jussien, and Éric Pinson, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 7298 of *Lectures Notes in Computer Science*, pages 211–227. Springer, 2012.

[HB12b] Stefan Heinz and J. Christopher Beck. Reconsidering mixed integer programming and MIP-based hybrids for scheduling. ZIB-Report 12-05, Zuse Institute Berlin, 2012.

[HKB13] Stefan Heinz, Wen-Yang Ku, and J. Christopher Beck. Recent improvements using constraint integer programming for resource allocation and scheduling. In Carla Gomes and Meinolf Sellmann, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 7874 of *Lecture Notes in Computer Science*, pages 12–27. Springer, 2013.

[HO03] John N. Hooker and Greger Ottosson. Logic-based Benders decomposition. *Mathematical Programming*, 96:33–60, 2003.

[Hoo00] John N. Hooker. *Logic-based Methods for Optimization.* Wiley, 2000.

[Hoo04] John N. Hooker. A hybrid method for planning and scheduling. In Mark Wallace, editor, *Principles and Practice of Constraint Programming – CP 2004*, volume 3258 of *Lecture Notes in Computer Science*, pages 305–316, 2004.

[Hoo05a] John N. Hooker. A hybrid method for the planning and scheduling. *Constraints*, 10(4):385–401, 2005.

[Hoo05b] John N. Hooker. Planning and scheduling to minimize tardiness. In Peter van Beek, editor, *Principles and Practice of Constraint Programming – CP 2005*, volume 3709 of *LNCS*, pages 314–327. Springer, 2005.

[Hoo07] John N. Hooker. Planning and scheduling by logic-based benders decomposition. *Operations Research*, 55(3):588–602, 2007.

[Hoo11] John N. Hooker. *Integrated Methods for Optimization.* Springer Publishing Company, Incorporated, 2nd edition, 2011.

[HS11] Stefan Heinz and Jens Schulz. Explanations for the cumulative constraint: An experimental study. In Panos M. Pardalos and Steffen Rebennack, editors, *Experimental Algorithms*, volume 6630 of *Lecture Notes in Computer Science*, pages 400–409. Springer, 2011.

[HSB13] Stefan Heinz, Jens Schulz, and J. Christopher Beck. Using dual presolving reductions to reformulate cumulative constraints. *Constraints*, 18(2):166–201, 2013.

[HY95] John N. Hooker and Hong Yan. *Logic circuit verification by Benders decomposition*, chapter 15, pages 267–288. MIT Press, 1995.

[HY02] John N. Hooker and Hong Yan. A relaxation of the cumulative constraint. In Pascal Van Hentenryck, editor, *Principles and Practice of Constraint Programming - CP 2002*, volume 2470 of *Lecture Notes in Computer Science*, pages 686–691. Springer Berlin Heidelberg, 2002.

[JG01] Vipul Jain and Ignacio E. Grossmann. Algorithms for hybrid MILP/CP models for a class of optimization problems. *INFORMS Journal on Computing*, 13(4):258–276, 2001.

[JPMSM08] Ines Lynce João P. Marques-Silva and Sharad Malik. *Conflict-driven clause learning SAT solvers*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, chapter 4, pages 131–153. IOS Press, 2008.

[KAA$^+$11] Thorsten Koch, Tobias Achterberg, Erling Andersen, Oliver Bastert, Timo Berthold, Robert E. Bixby, Emilie Danna, Gerald Gamrath, Ambros M. Gleixner, Stefan Heinz, Andrea Lodi, Hans Mittelmann, Ted Ralphs, Domenico Salvagnin, Daniel E. Steffy, and Kati Wolter. MIPLIB 2010. *Mathematical Programming Computation*, 3(2):103–163, 2011.

[KALM11] Oumar Koné, Christian Artigues, Pierre Lopez, and Marcel Mongeau. Event-based MILP models for resource-constrained project scheduling problems. *Computers & Operations Research*, 38(1):3–13, 2011.

[KH06] Rainer Kolisch and Sönke Hartmann. Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European Journal of Operational Research*, 174(1):23–37, 2006.

[Kiz04] Zeynep Kiziltan. Symmetry breaking ordering constraints: Thesis. *AI Communications*, 17:167–169, August 2004.

[KS96] Rainer Kolisch and Arno Sprecher. PSPLIB – A project scheduling problem library. *European Journal of Operational Research*, 96:205–216, 1996.

[KS99] Robert Klein and Armin Scholl. Computing lower bounds by destructive improvement: An application to resource-constrained project scheduling. *European Journal of Operational Research*, 112(2):322–346, 1999.

[Lau78] Jean-Louis Laurière. A language and a program for stating and solving combinatorial problems. *Artificial Intelligence*, 10(1):29–127, 1978.

[LD60] Ailsa H. Land and Alison G. Doig. An automatic method for solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.

[LL82] Abdelkader Lahrichi and Jacques-Louis Lions. Scheduling: the notions of hump, compulsory parts and their use in cumulative problems. *Comptes Rendus de l'Académie des Sciences, Série 1, Mathématique*, 294(2):209–211, 1982.

[LW92] 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.

[MH08] Luc Mercier and Pascal Van Hentenryck. Edge finding for cumulative scheduling. *INFORMS Journal on Computing*, 20(1):143–153, 2008.

[MH11] Michela Milano and Pascal Van Hentenryck. *Hybrid Optimization - The Ten Years of CPAIOR*, volume 45 of *Springer Optimization and its Applications.* Springer, 2011.

[MMZ$^+$01] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Annual Design Automation Conference*, DAC '01, pages 530–535, 2001.

[MS96] P. Martin and D. B. Shmoys. A new approach to computing optimal schedules for the job shop scheduling problem. In *Proceedings of the Fifth Conference on Integer Programming and Combinatorial Optimization*, 1996.

[MSS99] João P. Marques-Silva and Karem A. Sakallah. Grasp: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999.

[MSSU03] Rolf H. Möhring, Andreas S. Schulz, Frederik Stork, and Marc Uetz. Solving project scheduling problems by minimum cut computations. *Management Science*, 49(3):330–350, 2003.

[MW01] Hugues Marchand and Laurence A. Wolsey. Aggregation and mixed integer rounding to solve MIPs. *Operations Research*, 49(3):363–371, 2001.

[NN94] Yuri Nesterov and Arkadi Nemirovski. *Interior-Point Polynomial Algorithms in Convex Programming.* Studies in Applied and Numerical Mathematics. Society for Industrial and Applied Mathematics, 1994.

[Nui94] Wilhelmus Petronella Maria Nuijten. *Time and Resource Constrained Scheduling: A Constraint Satisfaction Approach.* PhD thesis, Eindhoven University of Technology, 1994.

[NW88] George L. Nemhauser and Laurence A. Wolsey. *Integer and Combinatorial Optimization.* Wiley-Interscience, New York, NY, USA, 1988.

[OQ13] Pierre Ouellet and Claude-Guy Quimper. Time-table extended-edge-finding for the cumulative constraint. In Christian Schulte, editor, *Principles and Practice of Constraint Programming - 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings*, volume 8124 of *Lecture Notes in Computer Science*, pages 562–577, 2013.

[PCVG94] Claude Le Pape, Philippe Couronné, Didier Vergamini, and Vincent Gosselin. Time-versus-capacity compromises in project scheduling. 13th Workshop of the UK Planning Special Interest Group, 1994.

[PSP] PSPLib. Project Scheduling Problem LIBrary. `http://www.om-db.wi.tum.de/psplib/`.

[PWW69] A. Alan B. Pritsker, Lawrence J. Waiters, and Philip M. Wolfe. Multiproject scheduling with limited resources: A zero-one programming approach. *Management Science*, 16(1):93–108, 1969.

[PX94] Panos M. Pardalos and Jue Xue. The maximum clique problem. *Journal of Global Optimization*, 4(3):301–328, 1994.

[QS94] Maurice Queyranne and Andreas S. Schulz. Polyhedral approaches to machine scheduling. Technical Report 408/1994, TU Berlin, 1994.

[RBW06] Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence).* Elsevier Science Inc., New York, NY, USA, 2006.

[Sal14] Domenico Salvagnin. Detecting and exploiting permutation structures in MIPs. In Helmut Simonis, editor, *Integration of AI and OR Techniques in Constraint Programming: 11th International Conference, CPAIOR 2014, Cork, Ireland, May 19-23, 2014. Proceedings*, volume 8451 of *Lecture Notes in Computer Science*, pages 29–44. Springer, 2014.

[Sav94] Martin W. P. Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. *ORSA Journal on Computing*, 6:445–454, 1994.

[Sch12] Jens Schulz. *Hybrid Solving Techniques for Project Scheduling Problems*. PhD thesis, Technische Universität Berlin, 2012.

[SCI] SCIP. Solving Constraint Integer Programs. `http://scip.zib.de/`.

[Sco10] Joseph Scott. Filtering algorithms for discrete cumulative resources. Master's thesis, Institutionen för informationsteknologi, 2010.

[SFS13] Andreas Schutt, Thibaut Feydy, and Peter Stuckey. Explaining time-table-edge-finding propagation for the cumulative resource constraint. In Carla Gomes and Meinolf Sellmann, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2013)*, volume 7874 of *Lecture Notes in Computer Science*, pages 234–250, 2013.

[SFSW11] Andreas Schutt, Thibaut Feydy, Peter J. Stuckey, and Mark G. Wallace. Explaining the cumulative propagator. *Constraints*, 16(3):250–282, 2011.

[SFSW13] Andreas Schutt, Thibaut Feydy, Peter J. Stuckey, and Mark G. Wallace. Solving RCPSP/max by lazy clause generation. *Journal of Scheduling*, 16:273–289, 2013.

[SS77] Richard M. Stallman and Gerald J. Sussman. Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence*, 9(2):135–196, 1977.

[SS06] Tuomas Sandholm and Rob Shields. Nogood learning for mixed integer programming. CMU-Report CMU-CS-06-155, Carnegie Mellon University, 2006.

[SW10] Andreas Schutt and Armin Wolf. A new $\mathcal{O}(n^2 \log n)$ not-first/not-last pruning algorithm for cumulative resource constraints. In David Cohen, editor, *Principles and Practice of Constraint Programming - CP 2010*, volume 6308 of *Lecture Notes in Computer Science*, pages 445–459, 2010.

[TG96] Metin Türkay and Ignacio E. Grossmann. Logic-based MINLP algorithms for the optimal synthesis of process networks. *Computers & Chemical Engineering*, 20(8):959–978, 1996.

[Tho01] E. S. Thorsteinsson. Branch-and-check: A hybrid framework integrating mixed integer programming and constraint logic programming. In *Proceedings of the Seventh International Conference on Principles and Practice of Constraint Programming (CP2001)*, pages 16–30. Springer, 2001.

[vH01] Willem Jan van Hoeve. The alldifferent constraint: A survey. *CoRR*, cs.PL/0105015, 2001.

[Vil05] Petr Vilím. Computing explanations for the unary resource constraint. In Roman Barták and Michela Milano, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Second International Conference, CPAIOR 2005, Prague, Czech Republic, May 30 - June 1, 2005*, volume 3524 of *Lecture Notes in Computer Science*, pages 396–409. Springer, 2005.

[Vil09a] Petr Vilím. Edge finding filtering algorithm for discrete cumulative resources in $O(kn \log n)$. In Ian P. Gent, editor, *Principles and Practice of Constraint Programming - CP 2009*, volume 5732 of *Lecture Notes in Computer Science*, pages 802–816, 2009.

[Vil09b] Petr Vilím. Max energy filtering algorithm for discrete cumulative resources. In Willem-Jan Hoeve and John N. Hooker, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 5547 of *Lecture Notes in Computer Science*, pages 294–308, 2009.

[Vil11] Petr Vilím. Timetable edge finding filtering algorithm for discrete cumulative resources. In Tobias Achterberg and J. Christopher Beck, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 6697 of *Lecture Notes in Computer Science*, pages 230–245, 2011.

[VW10] François Vanderbeck and Laurence A. Wolsey. Reformulation and decomposition of integer programs. In Michael Jünger, Thomas M. Liebling, Denis Naddef, George L. Nemhauser, William R. Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A. Wolsey, editors, *50 Years of Integer Programming 1958-2008*, pages 431–502. 2010.

[YAH10] Tallys Yunes, Ionut D. Aron, and John N. Hooker. An integrated solver for optimization problems. *Operations Research*, 58(2):342–356, 2010.

[ZMMM01] Lintao Zhang, Conor F. Madigan, Matthew W. Moskewicz, and Sharad Malik. Efficient conflict driven learning in boolean satisfiability solver. In *Proceedings of the 2001 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2001, San Jose, CA, USA, November 4-8, 2001*, pages 279–285, 2001.

[ZS96] Hantao Zhang and Mark E. Stickel. An efficient algorithm for unit propagation. In *In Proceedings of the Fourth International Symposium on Artificial Intelligence and Mathematics (AI-MATH'96), Fort Lauderdale (Florida USA*, pages 166–169, 1996.