Noname manuscript No.
(will be inserted by the editor)

Using dual presolving reductions to reformulate cumulative constraints

Stefan Heinz · Jens Schulz · J. Christopher Beck

the date of receipt and acceptance should be inserted later

Abstract Dual presolving reductions are a class of reformulation techniques that remove feasible or even optimal solutions while guaranteeing that at least one optimal solution remains, as long as the original problem was feasible. Presolving and dual reductions are important components of state-of-the-art mixed-integer linear programming solvers. In this paper, we introduce them both as unified, practical concepts in constraint programming solvers. Building on the existing idea of variable locks, we formally define and justify the use of dual information for cumulative constraints during a presolving phase of a solver. In particular, variable locks are used to decompose cumulative constraints, detect irrelevant variables, and infer variable assignments and domain reductions. Since the computational complexity of propagation algorithms typically depends on the number of variables and/or domain size, such dual reductions are a source of potential computational speed-up. Through experimental evidence on resource constrained project scheduling problems, we demonstrate that the conditions for dual reductions are present in well-known benchmark instances and that a substantial proportion of them can be solved to optimality in presolving – without search. While we consider this result very promising, we do not observe significant change in overall run-time from the use of our novel dual reductions.

Supported by the DFG Research Center MATHEON *Mathematics for key technologies* in Berlin.

Stefan Heinz
Zuse Institute Berlin
Takustr. 7, 14195 Berlin, Germany
E-mail: heinz@zib.de

Jens Schulz
Technische Universität Berlin, Institut für Mathematik
Straße des 17. Juni 136, 10623 Berlin, Germany
E-mail: jschulz@math.tu-berlin.de

J. Christopher Beck
Department of Mechanical & Industrial Engineering
University of Toronto, Toronto, Ontario M5S 3G8, Canada
E-mail: jcb@mie.utoronto.ca

Keywords dual reductions · cumulative constraints · presolving · variable locks

1 Introduction

The practice of automatically reformulating and improving models, called *presolving*, is an important step in modern mixed-integer linear programming (MIP) solvers. Presolving takes place *before* the tree search starts and tries to reduce the size of the model by, for example, removing irrelevant information such as redundant constraints or already-assigned variables; by decomposing or reformulating constraints (e.g., tightening the bounds of the variables or strengthening coefficients of the constraints); and by extracting structural information such as cliques (sets of binary variables that must sum up to one) that can be used by branching heuristics or cutting plane generation. Presolving collects global structure information and transforms the given problem instance into an equivalent instance w.r.t. the optimal value, that is potentially easier to solve. In the operations research community, presolving has been shown to be an important ingredient for linear programming [1,2,3] and mixed-integer programming [4]. An overview of presolving techniques for mixed-integer linear programs is given in [5,6,7].

In contrast to standard constraint propagation, but similar to symmetry breaking, *dual reductions* are inference techniques that reformulate a feasible problem instance by removing feasible or even optimal solutions while guaranteeing that at least one optimal solution remains.

Achterberg [5] defined a mechanism to implement dual reductions based on global information called *variable locks*. 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.

Our thesis in this paper is two-fold:

1. The use of presolving and the concept of dual reductions are valuable components of automated problem reformulation in constraint programming (CP).
2. Variable locks form a practical and useful source of global information upon which dual reductions in CP can be based.

To support this thesis, we present a straightforward generalization of variable locks for constraint programming, in particular for constraint optimization problems (COP). We define the property of a constraint to be *monotone in a variable* and show how this property can be used by a constraint to infer that it does not have to place variable locks on some of its variables. In our study, we consider the cumulative constraint for resource-constrained project scheduling problems, which, in general, must place locks on each variable in its scope. We formalize and prove several conditions under which the cumulative constraint can use global variable lock information to perform the following dual reductions: (i) decompose a cumulative constraint into two or more smaller cumulative constraints, (ii) remove a variable from the scope of a cumulative constraint, (iii) assign a variable, and (iv) remove a value from a variable domain. We implement these dual reductions, together with a scheduling-specific primal heuristic, as part of the presolving phase of the SCIP constraint integer

programming solver [8]. Our extensive computational results on resource constraint project scheduling problems convincingly demonstrate that a substantial proportion of a well-known benchmark set can be reformulated and often solved in presolving. Despite these positive results, however, our novel techniques do not result in a substantial change in the mean time to solve the benchmarks.

The paper is organized as follows. In the next section, we provide the necessary background for our contributions in this paper and discuss the relationship between presolving, variable locks, and dual reductions to techniques such as symmetry breaking and the use of dominance rules. In Section 3, we generalize the variable locks for COPs by introducing the property of a constraint to be monotone in a variable. Section 4 is devoted to the cumulative constraint and the formulation and formal analysis of our novel dual reductions. Computational results on resource constrained project scheduling problems are presented in Section 5. In Section 6, we discuss our results before concluding in Section 7.

2 Background

In this section, we review the concepts of presolving, primal vs. dual information, variable locks, and the relation between the work presented in this paper and existing work on symmetry breaking and dominance rules.

2.1 Presolving

As described in the introduction, presolving is a well-developed phase of problem solving for state-of-the-art MIP solvers. The basic idea is that before the tree search begins, computational effort is spent to reformulate the model to a smaller, simpler form, to employ heuristics to find feasible primal solutions, and to gather global information that can be used in the subsequent tree search.

In CP, while there has been extensive work in problem remodeling,[1] the majority of the work focuses on manual remodeling. Automated and semi-automated approaches have looked at language-level transformations such as translation to a constraint model [9, 10, 11] and decomposed representations of global constraints in solvers that do not support them [12]; the interpretation of richer annotated models [13, 14]; or symmetry breaking (discussed in depth in Section 2.3).

We believe that the introduction of presolving as a generic phase of CP search will provide a unifying framework, as some of this previous work can be implemented within presolving. Further, specifically identifying a presolving phase immediately prompts research into CP solving techniques that adapt and hybridize MIP presolving techniques.

Presolving may be particularly relevant to CP given the decomposed nature of the knowledge representation embodied by global constraints. While the modularity of global constraints has well-established strengths, it is precisely global information

[1] For example, the work that has appeared over the last decade at the *Workshops on Constraint Modelling and Reformulation*.

and higher-level structure that cannot be represented due to the limited channels of communication (i.e., variable domains) in standard CP. We speculate that efficient collection, representation, and exploitation of richer information as a standard solver phase is a promising direction to increase the power of CP solvers.[2]

2.2 Primal vs. dual information

The operations research literature has distinguished two types of presolving reductions: *primal* and *dual*. These terms stem from the duality theory for linear programs [16] where each reduction in the primal linear program has a counterpart in the corresponding dual linear program and vice versa. A solution that is primal and dual feasible is an optimal solution for both linear programs. For other optimization approaches, these terms have been generalized and are often used to distinguish between reductions that remove infeasible assignments (primal) and those that remove (potentially) feasible but dominated assignments (dual). Primal reductions are the usual style of inference in CP, based on proving infeasibility for some variable assignments. In contrast, dynamic symmetry breaking techniques are dual reductions.

2.3 Symmetry breaking and dominance rules

Unlike standard constraint propagation, symmetry based techniques remove feasible or optimal solutions while preserving at least one feasible or optimal solution. Symmetries therefore can be understood as a kind of dual information. More generally, a valid dominance rule guarantees the following: if there exists a solution, s_A, with characteristic A, then there must also exists a solution, s_B, with characteristic B and s_B is *as good as or better than* solution s_A. Therefore, the problem can be reformulated by removing any solution with characteristic A from the search space.

Much of the substantial body of work in CP on symmetry breaking (e.g., [17]) and on dominance rules (e.g., [18]) introduces problem specific symmetry breaking or dominance constraints. Automated detection of symmetries can also be done [19], albeit with substantial computational expense. Recently an approach was sketched [20], how dominance rules can be discovered in systematic way.

We view automatic symmetry detection techniques as promising sources of dual information in presolving. The computational expense for detection, the extent to which the symmetries exist, and the expense of breaking them will determine whether particular approaches can be successfully used in a general CP solver.

Instead of symmetries as a source of dual information, we choose here to explore presolving based on a weakened concept of constraint monotonicity and variable locks, a mechanism for gathering information about this monotonicity. Our reasoning here is that variable locks have a negligible computational overhead and have been shown to provide substantial benefit in solving mixed integer linear programs within a constraint-based system.

[2] Search heuristics based on aggregate solution counting can be seen as an example of this direction [15].

2.4 Variable locks

In most MIP solvers, dual reductions rely on the fact that it is easy to access the set of constraints that restrict a given variable. The constraints are often viewed as the rows of a matrix and the set of constraints relevant to a particular variable is exactly those rows with a non-zero entry (i.e., coefficient) in the column that represents the variable. This is called the *column representation*. Unfortunately, such a representation is, in general, not available in constraint-based systems due to the more complex semantics of a richer constraint language.

Variable locks were introduced to partially overcome this representational problem [5]. A constraint must declare a down-lock (resp. up-lock) on a variable if there exists a feasible assignment such that reducing (resp. increasing) the value of that variable, while leaving all other variables in the constraint scope unchanged, may violate the constraint. For example, we are given two integer variables $x_1, x_2 \in \mathbb{Z}$ and the following linear constraint:

$$5\,x_1 - 6\,x_2 \leq 8.$$

This constraint places an up-lock on x_1 and a down-lock on x_2 but, for example, no down-lock is placed on x_1 because reducing its value cannot change the constraint from feasible to infeasible. In the standard implementation, each constraint places variable locks on a sub-set of the variables in its scope and each variable maintains a count of the number of constraints that may become violated if an assigned value is increased or decreased.

Figure 1 shows the presolving reductions for CPLEX 12.3 and SCIP 2.1.0 [8], a constraint-based solver using variable locks. For the comparison we choose the MIPLIB2010 benchmark set [21] and run both solvers with and without dual presolving. The figure shows that both solvers make heavy use of the dual information and that the variable locks are almost as informative as the column representation. While CPLEX finds more reductions than SCIP in few instances, it is not clear if the differences are related to the limitation of the variable locks. In case of the instance `ex9`, SCIP reduces the number of variables and constraints to almost 0% whereas CPLEX stays well above 50%.[3] Overall, both solvers use dual information to substantially reduce problem sizes. These results raise the question of whether it is also possible and useful to do similar presolving for CPs.

3 Collecting dual information for constraint programs

In this section, we generalize variable locks to constraint programming 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.

[3] In version 12.3 and earlier, CPLEX ends presolving earlier due to an internal work limit (Tobias Achterberg, personal communication, November 30, 2011).

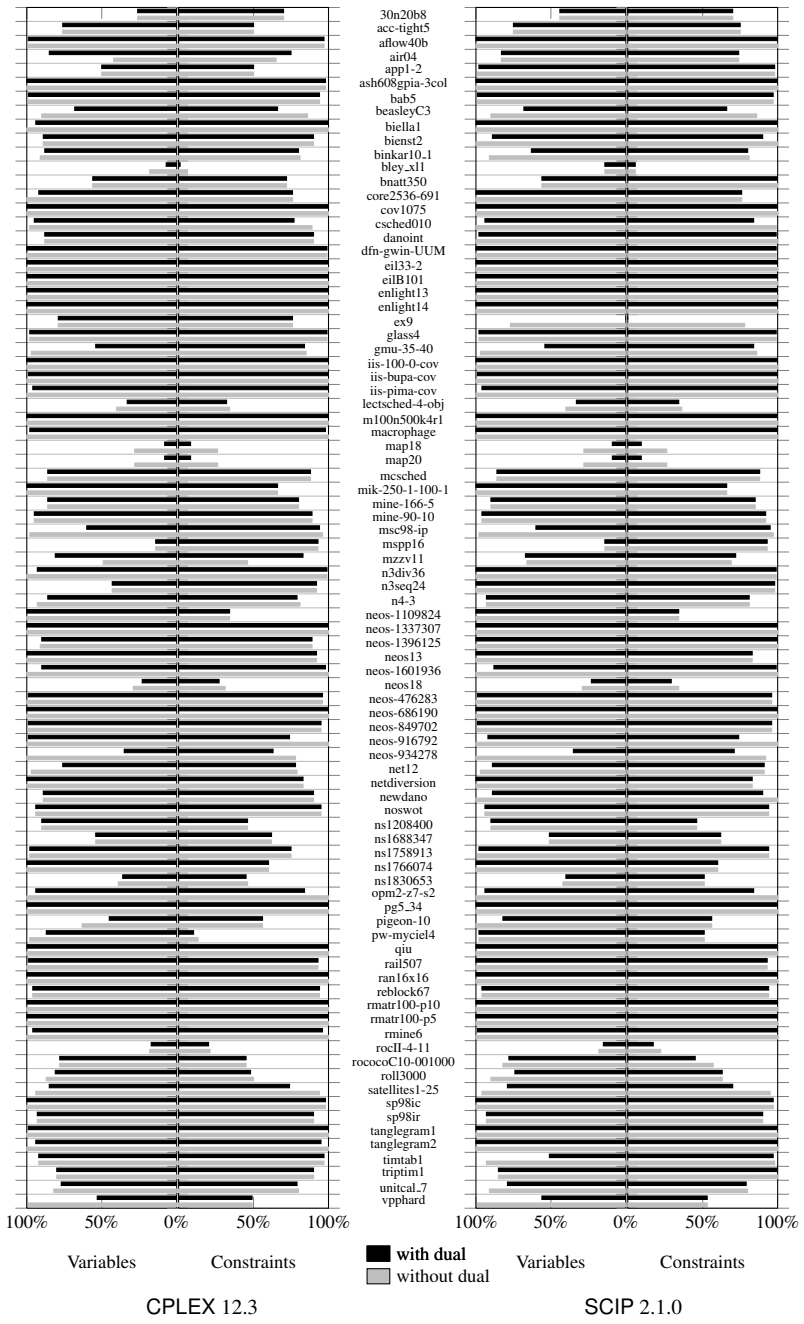


Fig. 1 Comparison of presolving reductions with and without dual information for all instances of the MIPLIB2010 benchmark set [21] using the MIP solver CPLEX 12.3 and the constraint-based solver SCIP 2.1.0 [8]. For each instance we depict the remaining percentage of variables and constraints after presolving.

Definition 1 A *constraint optimization problem* COP $= (X, \mathfrak{C}, \mathfrak{D}, f)$ consists in solving

$$c^\star = \min\{f(X) \mid \mathfrak{C}(X) = 1,\ X \in \mathfrak{D}\}$$

with $\mathfrak{D} = \mathfrak{D}_1 \times \cdots \times \mathfrak{D}_n$ 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 $X \in \mathfrak{D}$ with $x_j \in \mathfrak{D}_j$ 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. Before we introduce variable locks, we define the term *dual feasible* which describes dual reductions that are feasible for a COP.

Definition 2 Given a COP $= (X, \mathfrak{C}, \mathfrak{D}, f)$. For any variable x_j, a domain reduction $\mathfrak{D}'_j \subset \mathfrak{D}_j$ is called *dual feasible* if the non-reduced COP is infeasible or there exists an optimal solution to the COP where x_j is assigned to a value in $\mathfrak{D}'_j$.

Remark 1 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. [22]. Their conditions have to hold statically for each feasible (optimal) solution. For example, a variable x is *fixable* to a domain value d if, for any feasible (optimal) solution, an otherwise identical assignment that replaces the value of x by d is also a 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. However, most dual reductions known in MIP and the ones we introduce below can be mapped to one of the conditions introduced by Bordeaux et al. [22].

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). We assume here that, unless stated otherwise, all considered variables x_j have a totally ordered domain $\mathfrak{D}_j$.

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 in a variable*.

Definition 3 A constraint $\mathcal{C} : \mathfrak{D} \to \{0, 1\}$, is *monotone decreasing (increasing) in variable* x_j, if for all $\dot{x} \in \mathfrak{D}$ with $\mathcal{C}(\dot{x}) = 1$, it holds that $\mathcal{C}(\hat{x}) = 1$ for all $\hat{x} \in \mathfrak{D}$ with $\dot{x}_k = \hat{x}_k$ for all $k \neq j$ and $\hat{x}_j < \dot{x}_j$ ($\hat{x}_j > \dot{x}_j$).

If a constraint is either monotone decreasing or increasing in each variable in its scope, it is a *monotone constraint* (see Dechter [23]). Depending on the monotone status of a constraint, variable locks can be omitted.

Definition 4 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 *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 4 below).

Individual locks can be aggregated into dual information for a set of constraints. Here, following Achterberg [5], we accumulate locks by simply counting the number of constraints that down- or up-lock a variable, respectively. For a given constraint program, 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 Given four integer variables $x_1, x_2, x_3, x_4 \in \{0, \ldots, 10\}$ and the following linear constraint system:

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

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 $\zeta_4^- = 0$.

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 [5].

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 $\hat{x}_j \neq \min\{d \in \mathfrak{D}_j\}$, then it follows that the solutions $\dot{x}$ with $\dot{x}_k = \hat{x}_k$ for all $k \neq j$ and $\dot{x}_j \in \{d \in \mathfrak{D}_j \mid d < \hat{x}_j\}$ 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 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.[4] Such an inference is dual feasible and is called a *dual fixing*.

[4] 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.

As noted, using variable locks to detect such "half free" variables was already presented [5]. The following lemma summarizes this idea of dual fixing.

Lemma 1 *Given a COP* $= (X, \mathfrak{C}, \mathfrak{D}, f)$*. If a variable* x_j *with totally ordered domain* $\mathfrak{D}_j$ *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* $x_j = \min\{d \in \mathfrak{D}_j\}$ *(*$x_j = \max\{d \in \mathfrak{D}_j\}$*) is dual feasible.*

Example 2 Reconsider the linear constraints from Example 1. Given, additionally, an objective function $f(x) = x_1 + x_2 + x_3 + x_4$ 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 monotonically non-increasing in x_2.

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 analyzing Definition 4. 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 4, the result can 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 in monotonically non-decreasing or non-increasing, then Lemma 1 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 MIPLIB2010 [21]. For example the instances `bnatt350` contain isolated knapsack constraints which can be solved via dynamic programming.

4 The cumulative constraint

In this section, we discuss the use of the variable locks for the cumulative constraint. Given a finite set of jobs $\mathcal{J}$ and a capacity, $C \in \mathbb{N}$, the global cumulative constraint (as described in [24]) enforces that, at each point in time t, the total demand of the jobs running at t, does not exceed C. We restrict ourselves to the following settings: each job $j \in \mathcal{J}$ has a release date $\mathcal{R}_j \in \mathbb{N}$, a due date $\mathcal{D}_j \in \mathbb{N}$, a fixed non-negative processing time $p_j \in \mathbb{N}$, and a fixed non-negative resource demand $r_j \in \mathbb{N}$. Furthermore, C, is fixed and non-negative. The global cumulative constraint is given by

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

using vectors of start time (decision) variables $\boldsymbol{S}$, processing times $\boldsymbol{p}$, and demands $\boldsymbol{r}$, and the capacity. An assignment $\hat{\boldsymbol{S}}$, for the start time variables $\boldsymbol{S}$, is feasible if the following conditions hold:

$$\begin{aligned} &\hat{S}_j \in \{\mathcal{R}_j, \ldots, \mathcal{D}_j - p_j\} && \forall\, j \in \mathcal{J} \\ &\sum_{j \in \mathcal{J}} \mathbb{1}_{[\hat{S}_j, \hat{S}_j + p_j)}(t)\, r_j \leq C && \text{for all } t, \end{aligned} \tag{1}$$

where the indicator function is defined as $\mathbb{1}_M(x) = 1$ if $x \in M$, and zero otherwise.

Depending on the tightness of the *earliest start times*, $\mathrm{est}_j = \mathcal{R}_j$, *earliest completion times*, $\mathrm{ect}_j = \mathcal{R}_j + p_j$, *latest start times*, $\mathrm{lst}_j = \mathcal{D}_j - p_j$, and *latest completion times*, $\mathrm{lct}_j = \mathcal{D}_j$, for each job $j \in \mathcal{J}$, propagation algorithms (for example, see [25]) are able to update the release dates and due dates. The computational (worst case) complexity of these propagation algorithms depends on the number of jobs. Hence, removing jobs globally from the scope of a cumulative constraint results theoretically in a run-time improvement.

4.1 Effective horizon

The feasible time windows $[\mathrm{est}_j, \mathrm{lct}_j)$ of each job j can be used to define the first and last potential time points where the resource capacity can be possibly exceeded. We denote with hmin the minimum time point where the resource capacity could be exceeded and hmax the minimum time point where the resource capacity is surely satisfied. Formally:

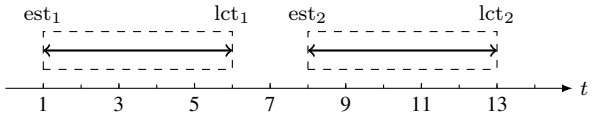
$$\begin{aligned} \mathrm{hmin} &= \inf\{t \in \mathbb{Z} \mid \sum_{j \in \mathcal{J}} \mathbb{1}_{[\mathrm{est}_j, \mathrm{lct}_j)}(t)\, r_j > C\} \\ \mathrm{hmax} &= \sup\{t \in \mathbb{Z} \mid \sum_{j \in \mathcal{J}} \mathbb{1}_{[\mathrm{est}_j, \mathrm{lct}_j)}(t-1)\, r_j > C\}. \end{aligned}$$

Note that in general $\mathrm{hmin} \leq \mathrm{hmax}$ does not hold. Using these two points in time the effective horizon can be defined.

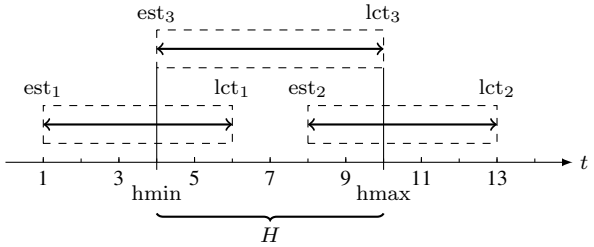
Definition 5 Given a set of jobs $\mathcal{J}$ with resource demands $\boldsymbol{r}$ that have to be scheduled on a resource with capacity of C. We define the *effective horizon* H as

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

If the effective horizon is empty, it follows from the definition of hmin and hmax that Condition (1) 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. The following example illustrates the effective horizon.



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



(b) Here $\mathrm{hmin} = 4$, $\mathrm{hmax} = 10$, and $H = [4, 10)$

Fig. 2 Illustration of effective horizon H for Example 3.

Example 3 Given are 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 2(a) depicts this situation. In this case $\mathrm{hmin} = \infty$ and $\mathrm{hmax} = -\infty$ 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 $H = [4, \ldots, 10)$ (see Figure 2(b)).

Figure 2(b) also illustrates the possibility of decomposing a cumulative constraint into two independent cumulative constraints. In case of the three jobs of Example 3, 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 the jobs 1 and 3 and the second jobs 2 and 3. The following lemma formalizes this decomposition.

Lemma 2 *Given are a set of jobs $\mathcal{J}$ with resource demands $\boldsymbol{r}$ that have to be scheduled on a resource with capacity of C. If there exists $t \in \{\mathrm{hmin}, \ldots, \mathrm{hmax} - 1\}$ such that*

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

then this resource restriction is decomposable into two resource restrictions by partitioning the set of jobs to $\mathcal{J}_1 = \{j \in \mathcal{J} \mid \mathrm{est}_j \leq t\}$ and $\mathcal{J}_2 = \{j \in \mathcal{J} \mid \mathrm{lct}_j \geq t\}$.

In general, the sets $\mathcal{J}_1$ and $\mathcal{J}_2$ are not disjoint.

4.2 Exploiting variable locks in the cumulative constraint

In general, a cumulative constraint must lock each start time variable in both directions since shifting a job in any direction may result in infeasibility. In such a case,

none of the dual reductions described in the previous section can be applied for these variables since the locks are strictly greater than zero. Therefore, we define and justify a situation where a cumulative constraint contributes to the variable locks by omitting some of them since it is monotone decreasing or increasing in a start time variable S_j. We restrict ourselves to the down-locks. All results stated can be symmetrically transformed to the up-locks.

We start by detecting *irrelevant* jobs ("completely free" start time variables). A job is called irrelevant for a 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. 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. The following two lemmas state this situation formally.

Lemma 3 *Given a cumulative constraint $\mathcal{C} = (\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}, C)$. A start time variable S_j with corresponding demand $r_j \leq C$ does not need to be locked in any direction if* $\mathrm{lct}_j \leq \mathrm{hmin}$, $\mathrm{est}_j \geq \mathrm{hmax}$, $r_j = 0$, *or* $p_j = 0$.

Proof The first two cases follow directly from the definition of hmin and hmax. The last two cases are obvious since the corresponding job does not require any resource capacity. □

The previous lemma considers the situation where a job is never processed within the effective horizon. The following lemma defines the case where a job must be processed through-out the effective horizon.

Lemma 4 *Given a cumulative constraint $\mathcal{C} = (\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}, C)$, a start time variable S_j with corresponding demand $r_j \leq C$ does not need to be locked in any direction if* $\mathrm{lst}_j \leq \mathrm{hmin}$ *and* $\mathrm{ect}_j \geq \mathrm{hmax}$.

Proof The conditions for the start time variables state that the corresponding job must start before the beginning of the effective horizon and must end after the end of the effective horizon. Therefore, the actual start time assigned does not impact the feasibility of any other variable assignments. Hence, no locking is required. □

Note that, in both cases the cumulative constraint is monotone decreasing and increasing in variable S_j. In case of Lemma 3 the corresponding start time variable can be removed from the scope of the cumulative constraint without any further adjustment. For Lemma 4, however, the capacity of the corresponding cumulative constraint needs to be decreased by the demand of the removed job.

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 $\mathrm{lst}_j \leq \mathrm{hmin}$. Depending on its processing time, this job could run either completely before the time window or overlapping the time window. In the latter case, we know that it overlaps with time window in such a way that hmin is included. See Figure 3 for an illustration. In this case moving the start time of this job earlier, out of the effective horizon H will be always feasible w.r.t. this constraint. Therefore, the cumulative constraint can omit the down-lock since it is monotone decreasing w.r.t. that start time variable.

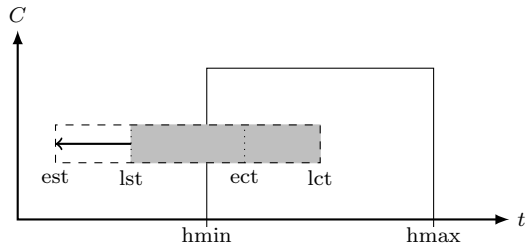


Fig. 3 Illustration of Lemma 5. Shifting the corresponding job earlier would only relax the situation within the effective horizon $H = [\mathrm{hmin}, \mathrm{hmax})$.

Lemma 5 *Given a cumulative constraint $\mathcal{C} = (\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}, C)$. If $\mathrm{lst}_j \leq \mathrm{hmin}$, then constraint $\mathcal{C}$ is monotone decreasing in start time variable S_j.*

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 $\mathrm{lst}_j = \mathcal{D}_j - p_j \leq \mathrm{hmin}$. 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 (i) $\boldsymbol{S}^1$ being a feasible assignment for $\mathcal{C}$, (ii) $S_i^1 = S_i^2$ if $i \neq j$, and (iii) $S_j^1 > S_j^2$, follows that $\boldsymbol{S}^2$ is a feasible assignment.

Per definition of the assignments for the start time variables, S_i^2 is a feasible (partial) solution for the cumulative constraint $\mathcal{C}$ for all $i \neq j$. In the case $S_j^2 < \mathrm{hmin} - p_j$, we know by definition of hmin that $\boldsymbol{S}^2$ is a feasible assignment. Assume $S_j^2 \geq \mathrm{hmin} - p_j$. Since $\mathrm{lst}_j \leq \mathrm{hmin}$ it follows that $S_j^2 < S_j^1 \leq \mathrm{hmin}$. Due to the last assumption we know that $\mathrm{hmin} \leq S_j^2 + p_j < S_j^1 + p_j$. Therefore, in both assignments, job j does not start later than hmin. For all $t \in [\mathrm{hmin}, \min\{S_j^2 + p_j, \mathrm{hmax} + 1\})$ we know that job j is also processed using the assignment $\boldsymbol{S}^1$. Hence, $\boldsymbol{S^2}$ is a feasible assignment. □

4.3 Dual reductions via variable locks

In the previous section, we stated conditions under which the cumulative constraint is monotone decreasing or increasing w.r.t. a start time variable 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.

The variable locks define the number of constraints that "block" the shifting of a certain variable to its lower or upper bound. A constraint can easily detect if it is the only one locking a certain variable if $\zeta^- = 1$ in case of the down-locks. Within the cumulative constraint, this information can be used to dual fix certain start time variables. The idea is that if the cumulative constraint is the only one locking the start time variable down (since $\mathrm{lst} > \mathrm{hmin}$), the best bound 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, then this variable can be dual fixed to its lower bound. Figure 4 illustrates this situation.

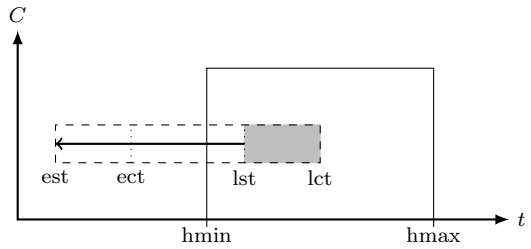


Fig. 4 Illustration of Lemma 6. Fixing the start time variable to its earliest start time, est, results in a situation where this job does not influence the effective horizon H.

Lemma 6 *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 est_j is dual feasible if the following conditions are satisfied:*

(i) $\mathrm{ect}_j \leq \mathrm{hmin}$,
(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 does not influence the feasibility 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 is the best thing to do w.r.t. the objective function.

Hence, if the problem is feasible, there exists an optimal solution with $S_j = \mathrm{est}_j$.
□

After fixing the variable to its earliest start time it follows that $\mathrm{lct}_j \leq \mathrm{hmin}$. Hence this job is irrelevant and can be removed from the constraint (see Lemma 3).

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. In such a situation, it is not possible to fix the start time variable but it is dual feasible to remove some values from the domain of the start time variable. Figure 5 depicts that statement and the following lemma formalizes it.

Lemma 7 *Given a cumulative constraint $\mathcal{C} = (\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}, C)$ and a start time variable S_j. Removing the values $\{\mathrm{est}_j + 1, \ldots, \mathrm{hmin}\}$ from the domain of a start time variable with demand $r_j \leq C$ is dual feasible if the following conditions are satisfied:*

(i) $\mathrm{est}_j < \mathrm{hmin}$,
(ii) only the cumulative constraint $\mathcal{C}$ down-locks S_j, and
(iii) the objective function f is monotonically non-decreasing in S_j.

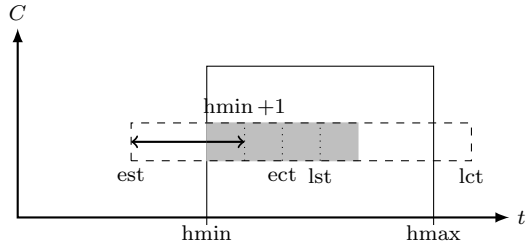


Fig. 5 Illustration of Lemma 7. Fixing the start time variable to a value greater than the earliest start time and smaller than or equal to hmin is dual dominated by fixing the start time variable to its earliest start time.

Proof Given a cumulative constraint $\mathcal{C} = (\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}, C)$ and a start time variable S_j that satisfies the conditions of the lemma. If the proposed domain reduction does not remove any feasible solution of constraint $\mathcal{C}$, it is obviously valid. Therefore, assume there exists an assignment $\boldsymbol{S}$ with $S_j \in \{\mathrm{est}_j + 1, \ldots, \mathrm{hmin}\}$ that is feasible for $\mathcal{C}$. It follows that the assignment $\boldsymbol{S}'$ with $S'_i = S_i$ for $i \neq j$ and $S'_j = \mathrm{est}_j$ is also feasible for $\mathcal{C}$. Due to condition (ii), 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 (iii) ensures that this shift does not increase the objective value. Hence, the proposed domain reduction is dual feasible. □

The previous two lemmas consider the case of a single cumulative constraint. Both lemmas are easily generalized for a set of cumulative constraints. For completeness we state these two corollaries. We denote with $\mathrm{hmin}(\mathcal{C})$ the first potential violated time point for a given cumulative constraint $\mathcal{C}$.

Corollary 2 *Given a set $\mathfrak{C}$ of cumulative constraints that have a start time variable S_j in their scope. Fixing the start time variable to its earliest start time est_j is dual feasible if the following conditions are satisfied:*

(i) $\mathrm{ect}_j \leq \mathrm{hmin}(\mathcal{C})$ *for all* $\mathcal{C} \in \mathfrak{C}$,
(ii) only the 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 *Given a set $\mathfrak{C}$ of cumulative constraints that have a start time variable S_j in their scope. Removing the values $\{\mathrm{est}_j + 1, \ldots, \min_{\mathcal{C} \in \mathfrak{C}} \mathrm{hmin}(\mathcal{C})\}$ from the domain of this variable is dual feasible if the following conditions are satisfied:*

(i) $\mathrm{est}_j \leq \mathrm{hmin}(\mathcal{C})$ *for all* $\mathcal{C} \in \mathfrak{C}$,
(ii) only the cumulative constraints $\mathcal{C} \in \mathfrak{C}$ *down-lock* S_j*, and*
(iii) the objective function f *is monotonically non-decreasing in* S_j.

4.4 Summary

All theoretical results presented in this section are related to jobs that are processed near the boundary of the effective horizon H. Overall, if a start time variable S_j is

only down-locked by one cumulative constraint $\mathcal{C}$, the objective function f is monotonically non-decreasing in S_j, and the feasible time window of job j is overlapping with hmin, then at least one of the above lemmas is applicable.

The effectiveness of these results w.r.t. the remaining problem can be sorted in the following order. Lemma 3 and 4 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 5 which states a condition such that one of the two locks (down or up) can be omitted. Finally, the weakest result is given by Lemma 6 and 7 which require several additional conditions to be applicable.

5 Computational results

In this section, we present computational results showing the impact of the dual reductions for the cumulative constraint.

5.1 Test sets

For testing, we use resource-constrained project scheduling problems with generalized precedence constraints. We are given a set $\mathcal{J}$ of non-preemptable jobs and a set $\mathcal{K}$ of renewable resources. Each resource $k \in \mathcal{K}$ has bounded capacity $C_k \in \mathbb{N}$. Every job j has a processing time $p_j \in \mathbb{N}$ and resource demands $r_{jk} \in \mathbb{N} \cup \{0\}$ for each resource $k \in \mathcal{K}$. The start time S_j of a job is constrained by its predecessors, given by a precedence graph $D = (V, A)$ with $V \subseteq \mathcal{J}$ and the distance function $d : A \to \mathbb{Z}$. An arc $(i, j) \in A$ represents a "start-to-start" precedence relationship between two jobs, i.e., job j cannot start before d_{ij} time units after job i starts. In case $d_{ij} = p_i$, 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 capacity and precedence constraints, such that the latest completion time of all jobs is minimized. We use a simple, standard constraint programming model:

$$
\begin{aligned}
\min \quad & \max_{j \in \mathcal{J}} S_j + p_j & & \\
\text{subject to} \quad & S_i + d_{ij} \leq S_j & & \forall\, (i,j) \in A \\
& \texttt{cumulative}(\boldsymbol{S}, \boldsymbol{p}, \boldsymbol{r}_{.k}, C_k) & & \forall\, k \in \mathcal{K} \\
& S_j \in \mathbb{Z} & & \forall\, j \in \mathcal{J}.
\end{aligned}
$$

Note that the linear constraints that describe the precedence conditions only lock each of the two start time variables in one direction: for the predecessor, an up-lock and for the successor, a down-lock. The objective function is regular [26], that is, monotonically non-decreasing in the start time variables, and hence allows shifting start time variables down.

For our experiments, we consider two different test sets on which we expect the dual reductions to have differing impact.

5.1.1 RCPSP

We use resource-constrained project scheduling problems with standard precedence constraints. That is, for all $(i,j) \in A$, $d_{ij} = p_i$. 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 an attractive problem class for our dual reductions 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 in the PSPLIB [27]. 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 latter 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 [28,29,30] which have the same type precedence conditions. This test set contains 55 instances which are highly cumulative, meaning that the ratio of resource capacity to resource demand is large. 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.

5.1.2 RCPSP/max

We also consider resource-constrained project scheduling problems with generalized precedence constraints. Informally, such a constraint represents a minimum or a maximum time that must elapse between a pair of start time variables. 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 the problem instances in the PSPLIB [27]. 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 J10, J20, and J30 each with 270 instances and 10, 20, and 30 jobs. In total there are 2520 instances.

5.2 Experimental setup

For our experiments, we use the constraint integer programming solver SCIP [5,8]. This solver is a constraint-based system, using variable locks to collect dual information. Within SCIP, constraints are implemented by constraint handlers in a similar fashion to SIMPL [31]. The cumulative constraint handler provides a variety of propagation algorithms and linear relaxations (see [32,33,30]). We use the time-tabling propagation algorithm [34] and perform over-load checking via edge-finding [35]. This combination is known to be most effective for the resource-constrained project

scheduling problems. We run the solver as a pure CP solver. Therefore, we used the predefined CP parameter settings.[5] In particular, the default CP settings do not use any linear relaxations.

For our purposes, we extend the existing cumulative constraint handler. We implemented all presolving reductions except the domain reductions in Lemma 7 and Corollary 3. These two reductions cannot easily be implemented in the current version of SCIP because variable domains are realized via intervals and so domain holes cannot be represented. In addition to the default SCIP components, which are used with their default settings, we added a primal heuristic based on a fast list scheduling algorithm [36]. 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.[6]

To get an impression of the utility of the introduced dual presolving steps we perform several experiments, 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?

We have three experimental conditions corresponding to our expectations of worst, reasonable, and best-case situations for the dual reductions. For each condition, we solve the problem instances twice, once with and once without the dual reductions. Our conditions are as follows:

- NOPRIMAL: 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 and PACK test sets. Such wide domains reduce the opportunities to make inferences.
- DEFAULT: SCIP is run with its default primal heuristics plus our problem specific scheduling heuristic.
- BOUNDED: In addition to the DEFAULT settings, the start time domains are bounded by the objective value of the best known solution[7] for each instance. This condition substantially reduces the start time domains and increases the potential to find domain reductions during the presolving phase.

5.3 Computational environment

All experiments were run on Intel Xeon Core 2.66 GHz computers (in 64 bit mode) with 4 MB cache, running Linux, and 8 GB of main memory. We used SCIP version

[5] In the interactive shell of SCIP, the CP solver settings can be set (and viewed) using the command `set emphasis cpsolver`.

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

[7] For the RCPSP instances we used the ones given in the PSPLIB [27] whereas for the RCPSP/max instances we considered the newly presented primal solutions in [37].

3.0.0 plus some bug fixes.[8] We did not use any linear programming solvers since we using SCIP as pure CP solver (see Section 5.2).

5.4 Results

The results are presented in Tables 1–5. The first two tables state results w.r.t. the presolving phase. Tables 4 and 5 show the overall impact of the presolving steps.

For RCPSP/max instances, no primal solution is found for many instances in the presolving phase, hence, the results for the NOPRIMAL setting are similar to the DEFAULT setting.

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 are applicable.

In the following, we analyze these results in more detail.

5.4.1 Applicability of dual reductions

Table 1 presents information about the number of inferences made by each implemented presolving reduction in the corresponding condition. The first column "Test set" states the name of test set. The remaining columns display the following information for each setting, that is NOPRIMAL, 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 sets, including the PACK instances. There are, therefore, 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 were 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 condition 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.

Decomposition. In case of the decomposition (Lemma 2), 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 time-tabling algorithms. Shrinking the feasible time window of jobs further increases the applicability of other

[8] These fixes will be available in SCIP 3.0.1.

Table 1 This table summarizes the appearance of the different presolving reductions. More detailed results are given in the appendix (Tables 6 and 7).

	NOPRIMAL					DEFAULT					BOUNDED				
Test set	Inst	Total	Max	Avg	Var	Inst	Total	Max	Avg	Var	Inst	Total	Max	Avg	Var
Constraint decompositions (Lemma 2)															
RCPSP	0.0%	–	–	–	–	8.1%	384	14	2.3	1.9	10.8%	706	16	3.1	2.8
RCPSP/max	0.0%	–	–	–	–	0.0%	–	–	–	–	16.6%	4633	185	11.1	16.6
Irrelevant variables due to no overlap with the effective horizon (Lemma 3)															
RCPSP	59.9%	47506	308	37.9	62.4	66.1%	52521	308	37.9	60.8	67.4%	59397	486	42.1	65.1
RCPSP/max	19.2%	8798	181	18.2	26.0	19.2%	8800	181	18.1	26.0	34.8%	159423	23802	182.0	963.9
Irrelevant variables due to an overlap with the effective horizon (Lemma 4)															
RCPSP	0.0%	–	–	–	–	1.5%	58	6	1.9	1.5	4.1%	237	19	2.8	3.2
RCPSP/max	0.0%	–	–	–	–	0.0%	–	–	–	–	4.3%	877	131	8.1	17.8
Variable lock adjustments (Lemma 5)															
RCPSP	0.0%	–	–	–	–	7.9%	578	18	3.5	3.5	13.0%	1778	68	6.5	7.9
RCPSP/max	0.0%	–	–	–	–	0.0%	–	–	–	–	10.5%	3612	348	13.6	34.0
Dual fixings due to a single constraint (Lemma 6)															
RCPSP	48.6%	9954	80	9.8	14.5	48.8%	10168	80	9.9	14.6	49.3%	10110	80	9.8	14.5
RCPSP/max	9.6%	503	15	2.1	2.0	9.6%	503	15	2.1	2.0	10.2%	527	15	2.1	2.0
Dual fixings due to a set of constraints (Corollary 2)															
RCPSP	64.1%	17377	87	12.9	20.6	65.4%	17487	87	12.8	20.4	66.1%	17503	87	12.6	20.3
RCPSP/max	19.2%	2721	45	5.6	7.5	19.2%	2722	45	5.6	7.5	20.4%	2806	46	5.4	7.3
All dual reductions															
RCPSP	85.1%	74837	393	42.0	74.0	85.7%	81196	393	45.2	74.3	86.5%	89731	538	49.5	78.1
RCPSP/max	21.5%	12022	225	22.2	33.0	21.5%	12025	225	22.1	32.9	37.0%	171878	23954	184.4	956.4

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 2 is, however, small compared to the total number of instances. For the RCPSP test set only 8% and 11% of the instances are affected in case of the DEFAULT and BOUNDED setting, respectively. For the RCPSP/max test set 16% of the instances are reformulated if the BOUNDED setting is applied.

Irrelevant jobs. Irrelevant jobs are those that run completely before or after the effective horizon (Lemma 3) or have to be processed during the entire effective horizon (Lemma 4). The first type of irrelevant jobs appears quite often for the RCPSP test set, independent of the experimental setting: on average up to 42 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. However, on instances where it is applicable, there is substantial inference; more, in fact, than on RCPSP instances. On one instance in the BOUNDED condition, 23802 reductions were triggered.

Knowing a good or even an optimal solution increases the number of identified 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 can the particular reductions 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 NOPRIMAL for the RCPSP test set and the DEFAULT setting for the RCPSP/max test set, where no decomposition takes place.

The second type of irrelevant jobs (Lemma 4) only arises (in these test sets) if a cumulative constraint is decomposed and can be seen as an inference arising from the decomposition.

Variable locks. The results for removing variable locks (Lemma 5) are similar to those for decomposition (Lemma 2). It is essential to have a good primal solution to trigger 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.

Dual fixings. The applicability of the dual fixing conditions (Lemma 6 and Corollary 2) seems to be (almost) independent of having a primal solution for both test sets. These two reductions are applicable quite often for the RCPSP test set. Note that within our implementation the single constraint dual fixing is done first, followed by the set-based version. As a consequence, the number of reductions stated for Corollary 2 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 sources 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 in almost half of the RCPSP test set and fixes on average 9 of the variables. For RCPSP/max, reductions are found in 10% of instances with an average of 2 variable fixings. Performing dual fixings on sets of constraints further increases the number of fixed variables during presolving by more than 12 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 final point 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 correspond constraint handler for that constraint type. Integrated reasoning about sets of constraints of the same type is similar to what is done in SIMPL [31] but does not seem to have been otherwise investigated in the CP literature.

Table 2 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.

	NOPRIMAL					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	48.1%	973	31	4.2	4.3	17.5%	211	10	2.5	1.9
60 jobs	89.4%	8935	61	20.8	25.3	60.2%	1532	31	5.3	4.6	19.2%	295	18	3.2	2.9
90 jobs	95.0%	13197	91	28.9	37.4	65.2%	2130	73	6.8	6.9	21.7%	280	11	2.7	2.3
120 jobs	88.3%	2801	32	5.3	4.6	87.7%	2750	35	5.2	4.6	54.2%	1241	40	3.8	3.8
RCPSP	85.1%	29569	91	16.6	25.3	64.9%	7385	73	5.4	5.3	28.9%	2027	40	3.4	3.3

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 in the dual fixings table 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. Overall, the introduced presolving steps arise for 85% of the instances (independent of the setting) for the RCPSP test sets (except for the PACK instances). In particular, the number of irrelevant variables per constraint is on average 10 (recall that these instances have 4 cumulative constraints) and the number of dual fixings is more than 9.

As expected, for the RCPSP/max test set, the reductions are not as effective. Still these reductions do arise in 21% of the instances if a primal solution is omitted and 37% of the instances if an objective limit (BOUNDED) is given. In both cases, some variables are fixed in presolving.

5.4.2 Presolving impact

The introduced presolving steps are applicable quite often for the RCPSP test sets 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 2 and 3 present, for each test sub-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 and so these results test whether our new techniques actually result in more inference above what SCIP is already capable of in presolving. The columns in the table have the same meaning as in Table 1.

RCPSP. When no primal solution is available (NOPRIMAL), the new dual reduction propagators are able to shrink the size w.r.t. the number of unfixed variables 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 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.

Table 3 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. For the test sets UBO500 and UBO1000 no additional variables were fixed and so we omit these rows.

	NOPRIMAL					DEFAULT					BOUNDED				
Test set	Inst	Total	Max	Avg	Var	Inst	Total	Max	Avg	Var	Inst	Total	Max	Avg	Var
TESTSETC	46.9%	1890	101	7.5	10.5	46.9%	1790	59	7.1	8.6	5.4%	147	60	5.1	11.2
TESTSETD	39.8%	1336	60	6.2	8.5	39.8%	1336	60	6.2	8.5	7.0%	112	13	2.9	2.9
J10	9.3%	35	3	1.4	0.7	9.3%	35	3	1.4	0.7	6.7%	41	11	2.3	2.4
J20	5.2%	18	2	1.3	0.5	5.2%	18	2	1.3	0.5	5.9%	20	3	1.2	0.6
J30	5.6%	19	3	1.3	0.6	5.6%	19	3	1.3	0.6	4.1%	13	2	1.2	0.4
UBO10	8.9%	10	3	1.2	0.7	8.9%	10	3	1.2	0.7	8.9%	20	11	2.5	3.3
UBO100	1.1%	1	1	1.0	0.0	1.1%	1	1	1.0	0.0	5.6%	7	2	1.4	0.5
UBO20	8.9%	12	2	1.5	0.5	8.9%	12	2	1.5	0.5	7.8%	11	2	1.6	0.5
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
UBO50	1.1%	1	1	1.0	0.0	1.1%	1	1	1.0	0.0	2.2%	4	3	2.0	1.0
RCPSP/max	21.5%	3324	101	6.1	9.1	21.5%	3224	60	5.9	8.2	5.5%	385	60	2.8	5.7

However, if a primal solution is known, the number of instances with additional fixed variables after the presolving phase is much smaller compared to the case where no primal solution is known. The reason for this result can be seen in Table 4 (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 drastically increases the number of instances solved in presolving. 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 2 and 6 additional variables.

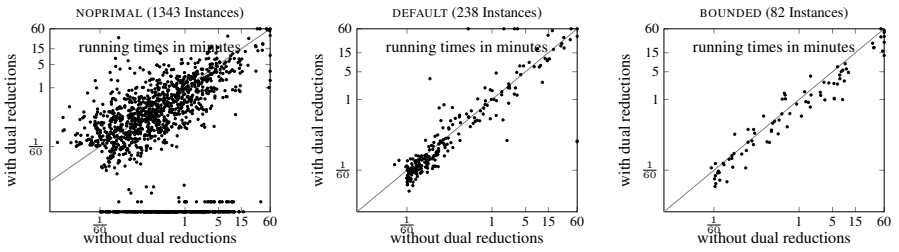
RCPSP/max. The impact of the dual reductions differs over the different test sub-sets. For TESTSETC and TESTSETD, we see a similar impact as for the RCPSP instances. With no primal solution (NOPRIMAL), 253 and 215 instances of TESTSETC and TESTSETD (each set contains 540 instances), respectively, are additionally reduced. On average 7 and 6 additional variables are fixed. When a primal solution is available, we observe the same phenomenon as for the RCPSP instances: the number of instances with an additional fixing decreases. For this test set the reason is not due to instances solved during presolving phase (see Table 5) 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 (25, 14, and 15, respectively) are affected when no primal solution is given. For these few instances only one variable is additionally fixed. When a primal bound is given, the number of affected instances decreases only slightly and the average number of fixed variables increases by a small amount for J10 and decreases by a small amount for the other two test sets.

For the UBO test sets (each of 90 instances) the impact is also negligible. Only between 0 and 8 instances are influenced by the dual reductions, independent of the setting. For these few instances, only 1 to 2 variables are additionally fixed.

Table 4 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) and omitting them (without dual) for the three different settings.

		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
NOPRIMAL	85.1%	1415	0	0	142	1055	1313	1372	1417	360	360	477	1186	1360	1393
DEFAULT	85.7%	1428	426	438	1199	1384	1408	1412	1425	428	441	1241	1385	1405	1412
BOUNDED	86.5%	1450	1192	1192	1374	1418	1430	1443	1454	1192	1192	1382	1420	1439	1444



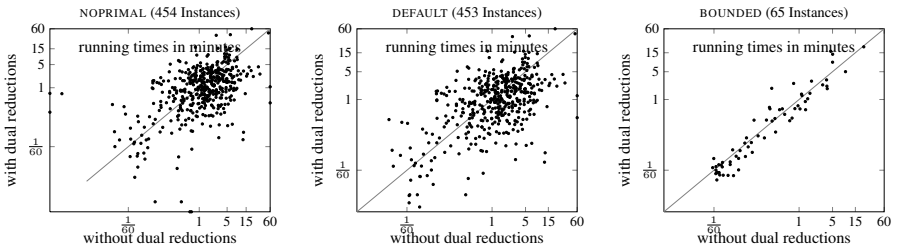
Summary. For both test sets, the tables show that the introduced presolving steps provide additional domain filtering: substantially more for the RCPSP test set than for RCPSP/max. Since the worst case complexity of the standard propagation algorithms and the subsequently required search depend on the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

5.4.3 Overall impact

In Tables 4 and 5, 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 1 hour for each instance. 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 where 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 in the root node process 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 figures. Hence, we omitted easy and unsolvable instances.

Table 5 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) and omitting them (without dual) for the three setting NOPRIMAL, DEFAULT, and BOUNDED.

		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
NOPRIMAL	21.5%	539	1	63	95	261	478	524	540	1	63	101	307	503	529
DEFAULT	21.5%	540	1	66	94	261	479	525	541	1	66	102	308	503	528
BOUNDED	37.0%	930	577	577	868	913	926	929	930	579	579	876	914	925	927



RCPSP. Table 4 shows the results for the 2095 instance of the RCPSP test set. It is notable that the dual reductions provide enough information that, in case of no primal solution (NOPRIMAL) is available, 360 instances are solved directly in the presolving phase. The performance figures are slightly in favor for the dual reductions. The numbers in the table show the same picture: using the dual settings solves a few instances more at each time step. Overall, 2 and 4 additional instances are solved within the time limit in case of the NOPRIMAL setting and BOUNDED setting. For the DEFAULT setting the dual setting fails to solve 3 instances that are solved when the dual reductions are omitted. From these results, it is not possible to conclude that either condition dominates.

RCPSP/max. The results for the overall impact of the dual reductions for this test set are given in Table 5. Here we get the same picture as for the RCPSP instances: the performance figures and numbers in the table are slightly in favor for the dual reductions. With respect to the number of solved instances, there is one more solved instance in case of the NOPRIMAL and DEFAULT settings.

Summary. Overall, a few instances are additionally solved within the time limit using the dual reductions. This number of instances, however, is meaningless compared to the size of the test sets. Overall, the impact of the dual reductions w.r.t. running time can be seen as slightly favorable, independent of the setting. This result is consistent with the fact that only few jobs (between 1 and 6) are additional 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 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 (NOPRIMAL), the dual reductions allow for solving 360 instances in the presolving phase that could not be solved in the presolving phase before.

6 Discussion

Variable locks and the effective horizon are two related concepts that can be exploited to create dual reduction techniques for cumulative constraints. Our first experiment shows that these reductions arise often for the analyzed 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 2 and 3). In particular, for RCPSP instances with 30, 60, and 90 jobs about 33% of the variables are additionally fixed, in contrast to not performing 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 6). This provides a theoretical speed-up as the worst-case complexity of standard propagation algorithms depends heavily on the number of jobs.

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 [38]. Similarly, Borrett and Tsang [39] observe that reformulating a model can have a negative impact depending on the algorithms used. The results in Tables 4 and 5, however, indicate that we are not seeing an overall increase in solving time for the investigated instances. Actually, a slight improvement can be observed.

Our results show that several instances of the RCPSP test set are easily solvable even without the presence of a primal solution. 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 critical to state-of-the-art performance. Our experimental results show that between 28% 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. For the RCPSP/max test set between 5% and 21% are further reformulated. We believe that these numbers alone indicate that presolving and dual techniques are an exciting direction and opportunity for constraint solving research.

7 Conclusions and outlook

Summary. Dual reductions are a form of problem reformulation that remove problem components (e.g. variables and/or values) and perhaps feasible and optimal solutions

while guaranteeing that at least a single optimal solution remains in the transformed search space (if the problem is feasible at all). While there has been work in 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 for implementation in a generic CP solver.

To concretize our approach, we build on the idea of variable locks, introduced by Achterberg [5] to allow the implementation of dual reductions for mixed integer programs within a constraint-based system. We formalize variable locks by defining constraints that are monotone decreasing or increasing in a variable (see Section 3).

Our main contribution (Section 4) is the proposal and analysis of the use of variable locks for dual reductions as part of the cumulative constraint. For that purpose, we defined the effective horizon which encapsulates the relevant time steps for a particular cumulative constraint. Based on variable locks and the effective horizon, we proved several conditions where a cumulative constraint can omit variable locks and, hence, provide dual information for the remaining constraints. We further presented results where these locks are used together with the structure of the cumulative constraint to (dual) fix decision variables.

In Section 5, we presented computational results using a well-known benchmark set of resource-constrained project scheduling problems. Our results demonstrated that the proposed reductions arise within these problem types. Furthermore, the dual reductions are capable of fixing variables that were not fixed by standard propagation algorithms during the presolving phase. However, for our test instances we did not observe any meaningful speed-up, though a slight improvement is observed in the overall performance.

Perspectives. One essential condition for the variable locks is that the variable domain is totally ordered. For problems without such an ordering, one could introduce such an order and apply the concepts of variable locks. Consider for example the all-different constraint [40,41]. For a given set of variables, it enforces that each variable takes a different value and, as a consequence, the domain of the variables need not be totally ordered. Introducing an order, however, can lead to dual fixings for even these types of variables. Consider the following example.

Example 4 Consider the variables x_1 with domain $\{1, 2, 4\}$ and x_2 with a domain of $\{1, 3, 4\}$. Using the natural ordering of the numbers, an all-different constraint over these two variables could not omit any lock. Changing these orders to $\{2, 1, 4\}$ for x_1 and $\{1, 4, 3\}$ for x_2, an all-different constraint over these two variables could omit the down-lock on variable x_1 and the up-lock on variable x_2.

The question arising from this observation is, which order on the domain values is the best one w.r.t. fixing variables. Or more generally, is it worthwhile to introduce value-based locks.

It is worth noting that the combination of variable locks and global constraints 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.

Conclusion. From the perspective of our thesis as stated in Section 1, we have demonstrated that the use of dual reductions in a presolving phase can lead to substantial problem reformulation and, furthermore, that variable locks provide an inexpensive mechanism to gather global dual information. Our experimental results indicate that we did not achieve a meaningful decrease in the mean time to solve the benchmark problem instances. Nevertheless, we believe that the novel use of presolving for CP opens promising research directions that will lead to stronger solver performance.

Acknowledgements We thank the anonymous referees for their extensive and valuable comments. We also thank Brahim Hnich for a discussion of monotone constraints.

References

1. Karwan, M.H., Lotfi, V., Telgen, J., Zionts, S.: Redundancy in mathematical programming. A state-of-the-art survey. Volume 206 of Lecture Notes in Economics and Mathematical Systems. Springer (1983)
2. Bixby, R.E., Wagner, D.K.: A note on detecting simple redundancies in linear systems. Operation Research Letters **6**(1) (1987) 15–17
3. Imbert, J.L., Hentenryck, P.V.: Redundancy elimination with a lexicographic solved form. Annals of Mathematics and Artificial Intelligence **17**(1–2) (1996) 85–106
4. Savelsbergh, M.W.P.: Preprocessing and probing techniques for mixed integer programming problems. ORSA Journal on Computing **6** (1994) 445–454
5. Achterberg, T.: Constraint Integer Programming. PhD thesis, Technische Universität Berlin (2007)
6. Guzelsoy, M.: Dual Methods in Mixed Integer Linear Programming. PhD thesis, Lehigh University,Industrial and Systems Engineering (2010)
7. Mahajan, A.: Presolving mixed-integer linear programs. Preprint ANL/MCS-P1752-0510, Mathematics and Computer Science Division (2010)
8. Achterberg, T.: Scip: Solving constraint integer programs. Mathematical Programming Computation **1**(1) (2009) 1–41
9. Gent, I.P., Miguel, I., Rendl, A.: Tailoring solver-independent constraint models: a case study with essence' and minion. In Miguel, I., Ruml, W., eds.: Abstraction, Reformulation, and Approximation. Volume 4612 of LNCS. (2007) 184–199
10. Frisch, A.M., Harvey, W., Jefferson, C., Martínez-Hernández, B., Miguel, I.: Essence: A constraint language for specifying combinatorial problems. Constraints **13** (2008) 268–306
11. Rendl, A.: Effective Compilation of Constraint Models. PhD thesis, University of St Andrews (2010)
12. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: Minizinc: Towards a standard cp modelling language. In Bessiere, C., ed.: Principles and Practice of Constraint Programming - CP 2007. Volume 4741 of LNCS., Springer (2007) 529–543
13. de la Banda, M.J.G., Marriott, K., Rafeh, R., Wallace, M.: The modelling language zinc. In Benhamou, F., ed.: Principles and Practice of Constraint Programming - CP 2006. Volume 4204 of LNCS. (2006) 700–705

14. Marriott, K., Nethercote, N., Rafeh, R., Stuckey, P.J., de la Banda, M.G., Wallace, M.: The design of the zinc modelling language. Constraints **13**(3) (2008) 229–267
15. Pesant, G., Quimper, C., Zanarini, A.: Counting-based search: Branching heuristics for constraint satisfaction problems. Journal of Artificial Intelligence Research **43** (2012) 173–210
16. Dantzig, G.B., Thapa, M.N.: Linear Programming 2. Springer Series in Operations Research. Springer (2003)
17. Gent, I.P., Petrie, K.E., Puget, J.F.: Symmetry in Constraint Programming. In: Handbooks of Constraint Programming. Elsevier (2006)
18. Prestwich, S.D., Beck, J.C.: Exploiting dominance in three symmetric problems. In: Proceedings of the Fourth International Workshop on Symmetry and Constraint Satisfaction Problems. (2004)
19. Puget, J.F.: Automatic detection of variable and value symmetries. In van Beek, P., ed.: Principles and Practice of Constraint Programming - CP 2005. Volume 3709 of LNCS. (2005) 475–489
20. Chu, G., Stuckey, P.J.: A generic method for identifying and exploiting dominance relations. In Milano, M., ed.: Principles and Practice of Constraint Programming - CP 2012. LNCS (2012) 6–22
21. Koch, T., Achterberg, T., Andersen, E., Bastert, O., Berthold, T., Bixby, R.E., Danna, E., Gamrath, G., Gleixner, A.M., Heinz, S., Lodi, A., Mittelmann, H., Ralphs, T., Salvagnin, D., Steffy, D.E., Wolter, K.: MIPLIB 2010. Mathematical Programming Computation **3**(2) (2011) 103–163
22. Bordeaux, L., Cadoli, M., Mancini, T.: A unifying framework for structural properties of csps: definitions, complexity, tractabilit. J. Artif. Int. Res. **32**(1) (June 2008) 607–629
23. Dechter, R.: Constraint processing. Elsevier Morgan Kaufmann (2003)
24. Aggoun, A., Beldiceanu, N.: Extending chip in order to solve complex scheduling and placement problems. Mathematical and Computer Modelling **17**(7) (1993) 57–73
25. Baptiste, P., Pape, C.L., Nuijten, W.: Constraint-based Scheduling. Kluwer Academic Publishers (2001)
26. Baptiste, P., Pape, C.L.: Scheduling a single machine to minimize a regular objective function under setup constraints. Discrete Optimization **2**(1) (2005) 83–99
27. PSPLib: Project Scheduling Problem LIBrary. `http://129.187.106.231/psplib/`
28. Baptiste, P., Pape, C.L.: Constraint propagation and decomposition techniques for highly disjunctive and highly cumulative project scheduling problems. Constraints **5**(1/2) (2000) 119–139
29. Artigues, C., Demassey, S., Néron, E., eds.: Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications. iSTE (2008)
30. Heinz, S., Schulz, J.: Explanations for the cumulative constraint: An experimental study. In Pardalos, P.M., Rebennack, S., eds.: Experimental Algorithms. Volume 6630 of LNCS., Springer (2011) 400–409
31. Yunes, T., Aron, I.D., Hooker, J.N.: An integrated solver for optimization problems. Operations Research **58**(2) (2010) 342–356
32. Berthold, T., Heinz, S., Lübbecke, M.E., Möhring, R.H., Schulz, J.: A constraint integer programming approach for resource-constrained project scheduling. In Lodi, A., Milano, M., Toth, P., eds.: Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. Volume 6140 of LNCS., Springer (2010) 313–317
33. Berthold, T., Heinz, S., Schulz, J.: An approximative criterion for the potential of energetic reasoning. In Marchetti-Spaccamela, A., Segal, M., eds.: Theory and Practice of Algorithms in (Computer) Systems. Volume 6595 of LNCS., Springer (2011) 229–239
34. Klein, R., Scholl, A.: Computing lower bounds by destructive improvement: An application to resource-constrained project scheduling. European Journal of Operational Research **112**(2) (1999) 322–346
35. Vilím, P.: Max energy filtering algorithm for discrete cumulative resources. In van Hoeve, W.J., Hooker, J.N., eds.: Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. Volume 5547 of LNCS. (2009) 294–308
36. Möhring, R.H., Schulz, A.S., Stork, F., Uetz, M.: Solving project scheduling problems by minimum cut computations. Management Science **49**(3) (2003) 330–350
37. Schutt, A., Feydy, T., Stuckey, P.J., Wallace, M.G.: Solving rcpsp/max by lazy clause generation. Journal of Scheduling (2012) accepted.
38. Kiziltan, Z.: Symmetry breaking ordering constraints: Thesis. AI Communications **17** (August 2004) 167–169
39. Borrett, J.E., Tsang, E.P.K.: A context for constraint satisfaction problem formulation selection. Constraints **6**(4) (2001) 299–327
40. Laurière, J.L.: A language and a program for stating and solving combinatorial problems. Artificial Intelligence **10**(1) (1978) 29–127
41. van Hoeve, W.J.: The alldifferent constraint: A survey. CoRR **cs.PL/0105015** (2001)

A Detailed computational results

In this section, we present detailed computational results that are helpful to justify our claims in Section 5. For each summary line in Table 1 we have included one table that shows how certain results arose within a particular test set. Table 6 presents the results for the test set of the RCPSP instances and Table 7, the results for the RCPSP/max test sets. The columns have the same meaning as in the summary table. The first column "Test set" states the name of test sub-set. The remaining columns display for each setting, that is NOPRIMAL, DEFAULT, and BOUNDED (see Section 5.2), the following information. The first column "Inst" shows the percentage of instances where at least one reduction was detected in the considered presolving step. The second column prints the "Total" number of times the presolving reduction was applied. The remaining columns display, for those instances were the reduction was applied at least once, the maximum ("Max") number of times it was applied for a single instance, the average ("Avg") number of times, and the standard deviation ("Var"). When no reduction was found for the whole test set we print "–". If this is the case for all three settings we omit the whole results line. Since no primal solution is available during the presolving phase for the RCPSP/max instances, the results of NOPRIMAL and DEFAULT are the same. Therefore, we omit the corresponding columns for the NOPRIMAL condition.

B Re-running the experiments

The results we presented w.r.t. the impact of the introduced dual reductions (Tables 1, 2, 3, 6, and 7) can be reproduced with the scheduler example of SCIP version 3.0.1 release. For getting the statistics of the different dual reductions, you need to define `SCIP_STATISTIC` in the source code of the cumulative constraint handler (`cons_cumulative.c`). The dual reductions can be turned off and on (default) via the Boolean parameter `constraints/cumulative/dualpresolve`.

Table 6 This table summarizes the appearance of the different presolving reductions for each test sub-set of the RCPSP instances separately. For the PACK test set none of the dual reductions was applicable there we omit these rows.

	NOPRIMAL					DEFAULT					BOUNDED				
Test set	Inst	Total	Max	Avg	Var	Inst	Total	Max	Avg	Var	Inst	Total	Max	Avg	Var
Constraint decompositions (Lemma 2)															
30 jobs	0.0%	–	–	–	–	14.8%	186	14	2.6	2.1	23.1%	355	16	3.2	3.0
60 jobs	0.0%	–	–	–	–	11.2%	131	11	2.4	2.0	10.4%	153	10	3.1	2.3
90 jobs	0.0%	–	–	–	–	7.5%	56	5	1.6	0.9	5.2%	77	11	3.1	2.5
120 jobs	0.0%	–	–	–	–	1.3%	11	2	1.4	0.5	6.8%	121	15	3.0	3.0
RCPSP	0.0%	–	–	–	–	8.1%	384	14	2.3	1.9	10.8%	706	16	3.1	2.8
Irrelevant variables due to no overlap with the effective horizon (Lemma 3)															
30 jobs	50.2%	6578	96	27.3	29.0	61.7%	7930	151	26.8	27.7	63.8%	10573	313	34.6	38.4
60 jobs	64.0%	14468	200	47.1	61.0	71.5%	16749	306	48.8	61.1	71.5%	16899	272	49.3	60.8
90 jobs	69.0%	22298	308	67.4	93.4	74.8%	23407	308	65.2	90.6	73.3%	23926	308	68.0	91.7
120 jobs	62.5%	4162	76	11.1	9.6	64.3%	4435	116	11.5	11.7	68.3%	7999	486	19.5	44.5
RCPSP	59.9%	47506	308	37.9	62.4	66.1%	52521	308	37.9	60.8	67.4%	59397	486	42.1	65.1
Irrelevant variables due to an overlap with the effective horizon (Lemma 4)															
30 jobs	0.0%	–	–	–	–	2.9%	21	3	1.5	0.6	8.8%	119	14	2.8	3.0
60 jobs	0.0%	–	–	–	–	2.5%	32	6	2.7	2.0	4.2%	51	19	2.5	3.9
90 jobs	0.0%	–	–	–	–	1.0%	5	1	1.0	0.0	2.3%	28	8	2.5	2.1
120 jobs	0.0%	–	–	–	–	0.0%	–	–	–	–	2.2%	39	13	3.0	3.4
RCPSP	0.0%	–	–	–	–	1.5%	58	6	1.9	1.5	4.1%	237	19	2.8	3.2
Variable lock adjustments (Lemma 5)															
30 jobs	0.0%	–	–	–	–	11.7%	169	13	3.0	2.7	24.6%	873	68	7.4	9.3
60 jobs	0.0%	–	–	–	–	10.8%	228	18	4.4	4.7	12.3%	385	57	6.5	8.1
90 jobs	0.0%	–	–	–	–	10.0%	166	14	3.5	2.9	6.0%	183	19	6.3	4.9
120 jobs	0.0%	–	–	–	–	1.5%	15	4	1.7	1.1	11.0%	337	24	5.1	5.3
RCPSP	0.0%	–	–	–	–	7.9%	578	18	3.5	3.5	13.0%	1778	68	6.5	7.9
Dual fixings due to a single constraint (Lemma 6)															
30 jobs	47.1%	1465	25	6.5	6.5	47.7%	1549	25	6.8	6.6	49.2%	1490	25	6.3	6.5
60 jobs	53.3%	2805	52	11.0	14.1	53.3%	2836	52	11.1	14.1	53.8%	2845	52	11.0	14.1
90 jobs	57.1%	4193	80	15.3	21.8	57.1%	4290	80	15.7	22.0	57.1%	4216	80	15.4	21.8
120 jobs	43.8%	1491	23	5.7	5.3	43.8%	1493	23	5.7	5.4	44.0%	1559	35	5.9	5.8
RCPSP	48.6%	9954	80	9.8	14.5	48.8%	10168	80	9.9	14.6	49.3%	10110	80	9.8	14.5
Dual fixings due to a set of constraints (Corollary 2)															
30 jobs	56.5%	2716	30	10.0	10.5	59.2%	2768	30	9.7	10.4	59.0%	2755	30	9.7	10.4
60 jobs	70.0%	5333	59	15.9	20.3	71.5%	5361	59	15.6	20.2	72.5%	5371	59	15.4	20.1
90 jobs	75.2%	8018	87	22.2	30.1	76.5%	8045	87	21.9	29.9	76.2%	8029	87	21.9	30.0
120 jobs	62.5%	1310	20	3.5	2.8	62.7%	1313	21	3.5	2.8	64.5%	1348	20	3.5	2.8
RCPSP	64.1%	17377	87	12.9	20.6	65.4%	17487	87	12.8	20.4	66.1%	17503	87	12.6	20.3
All dual reductions															
30 jobs	76.5%	10759	0	29.3	35.9	79.0%	12623	0	33.3	35.5	81.5%	16165	0	41.3	47.2
60 jobs	89.4%	22606	0	52.7	74.1	89.4%	25337	0	59.1	75.5	90.0%	25704	0	59.5	75.4
90 jobs	95.0%	34509	0	75.7	112.6	95.0%	35969	0	78.9	111.9	95.6%	36459	0	79.4	112.3
120 jobs	88.3%	6963	0	13.1	11.4	88.5%	7267	0	13.7	13.4	88.5%	11403	0	21.5	44.1
RCPSP	85.1%	74837	393	42.0	74.0	85.7%	81196	393	45.2	74.3	86.5%	89731	538	49.5	78.1

Table 7 This table summarizes the appearance of the different presolving reductions for each test sub-set of the RCPSP/max instances separately. If for a test sub-set a particular dual reductions did not arose we omit the corresponding row in the table.

	NOPRIMAL					DEFAULT					BOUNDED				
Test set	Inst	Total	Max	Avg	Var	Inst	Total	Max	Avg	Var	Inst	Total	Max	Avg	Var
Constraint decompositions (Lemma 2)															
TESTSETC	0.0%	–	–	–	–	0.0%	–	–	–	–	7.4%	623	107	15.6	20.9
TESTSETD	0.0%	–	–	–	–	0.0%	–	–	–	–	10.2%	1260	185	22.9	32.5
J10	0.0%	–	–	–	–	0.0%	–	–	–	–	28.9%	400	29	5.1	5.0
J20	0.0%	–	–	–	–	0.0%	–	–	–	–	25.9%	549	37	7.8	7.2
J30	0.0%	–	–	–	–	0.0%	–	–	–	–	19.3%	512	50	9.8	9.7
UBO10	0.0%	–	–	–	–	0.0%	–	–	–	–	44.4%	265	15	6.6	3.9
UBO100	0.0%	–	–	–	–	0.0%	–	–	–	–	18.9%	284	59	16.7	16.3
UBO20	0.0%	–	–	–	–	0.0%	–	–	–	–	33.3%	285	40	9.5	8.8
UBO200	0.0%	–	–	–	–	0.0%	–	–	–	–	11.1%	124	41	12.4	12.1
UBO50	0.0%	–	–	–	–	0.0%	–	–	–	–	26.7%	259	56	10.8	12.6
UBO500	0.0%	–	–	–	–	0.0%	–	–	–	–	3.3%	72	69	24.0	31.8
RCPSP/max	0.0%	–	–	–	–	0.0%	–	–	–	–	16.6%	4633	185	11.1	16.6
Irrelevant variables due to no overlap with the effective horizon (Lemma 3)															
TESTSETC	44.4%	5144	174	21.4	27.9	44.6%	5146	174	21.4	27.9	49.8%	28113	3581	104.5	350.6
TESTSETD	35.2%	3414	181	18.0	25.9	35.2%	3414	181	18.0	25.9	42.4%	53362	10894	233.0	852.8
J10	5.2%	74	14	5.3	3.2	5.2%	74	14	5.3	3.2	35.6%	2402	236	25.0	33.8
J20	3.7%	33	5	3.3	1.2	3.7%	33	5	3.3	1.2	28.9%	5536	443	71.0	85.6
J30	3.7%	37	5	3.7	1.2	3.7%	37	5	3.7	1.2	23.3%	7951	974	126.2	167.8
UBO10	8.9%	39	12	4.9	2.8	8.9%	39	12	4.9	2.8	52.2%	1521	87	32.4	25.4
UBO100	1.1%	2	2	2.0	0.0	1.1%	2	2	2.0	0.0	21.1%	13481	3146	709.5	788.5
UBO20	8.9%	45	10	5.6	2.3	8.9%	45	10	5.6	2.3	38.9%	2943	422	84.1	94.6
UBO200	2.2%	7	5	3.5	1.5	2.2%	7	5	3.5	1.5	12.2%	13294	4393	1208.5	1187.9
UBO50	1.1%	3	3	3.0	0.0	1.1%	3	3	3.0	0.0	28.9%	5908	1155	227.2	270.4
UBO500	0.0%	–	–	–	–	0.0%	–	–	–	–	3.3%	24912	23802	8304.0	10959.9
RCPSP/max	19.2%	8798	181	18.2	26.0	19.2%	8800	181	18.1	26.0	34.8%	159423	23802	182.0	963.9
Irrelevant variables due to an overlap with the effective horizon (Lemma 4)															
TESTSETC	0.0%	–	–	–	–	0.0%	–	–	–	–	3.0%	281	131	17.6	33.9
TESTSETD	0.0%	–	–	–	–	0.0%	–	–	–	–	4.3%	280	91	12.2	22.1
J10	0.0%	–	–	–	–	0.0%	–	–	–	–	6.7%	56	13	3.1	2.9
J20	0.0%	–	–	–	–	0.0%	–	–	–	–	4.4%	60	25	5.0	6.9
J30	0.0%	–	–	–	–	0.0%	–	–	–	–	4.1%	78	20	7.1	5.6
UBO10	0.0%	–	–	–	–	0.0%	–	–	–	–	10.0%	20	4	2.2	1.0
UBO100	0.0%	–	–	–	–	0.0%	–	–	–	–	3.3%	9	6	3.0	2.2
UBO20	0.0%	–	–	–	–	0.0%	–	–	–	–	6.7%	43	21	7.2	6.7
UBO200	0.0%	–	–	–	–	0.0%	–	–	–	–	2.2%	4	2	2.0	0.0
UBO50	0.0%	–	–	–	–	0.0%	–	–	–	–	6.7%	25	8	4.2	2.5
UBO500	0.0%	–	–	–	–	0.0%	–	–	–	–	2.2%	21	20	10.5	9.5
RCPSP/max	0.0%	–	–	–	–	0.0%	–	–	–	–	4.3%	877	131	8.1	17.8
Variable lock adjustments (Lemma 5)															
TESTSETC	0.0%	–	–	–	–	0.0%	–	–	–	–	5.6%	948	348	31.6	67.6
TESTSETD	0.0%	–	–	–	–	0.0%	–	–	–	–	7.6%	1196	297	29.2	54.1
J10	0.0%	–	–	–	–	0.0%	–	–	–	–	19.3%	238	27	4.6	5.1
J20	0.0%	–	–	–	–	0.0%	–	–	–	–	14.8%	293	67	7.3	11.5
J30	0.0%	–	–	–	–	0.0%	–	–	–	–	12.2%	369	58	11.2	13.8
UBO10	0.0%	–	–	–	–	0.0%	–	–	–	–	27.8%	103	16	4.1	3.7
UBO100	0.0%	–	–	–	–	0.0%	–	–	–	–	8.9%	61	24	7.6	8.9
UBO20	0.0%	–	–	–	–	0.0%	–	–	–	–	16.7%	131	44	8.7	10.9
UBO200	0.0%	–	–	–	–	0.0%	–	–	–	–	4.4%	31	12	7.8	3.2
UBO50	0.0%	–	–	–	–	0.0%	–	–	–	–	15.6%	164	52	11.7	14.0
UBO500	0.0%	–	–	–	–	0.0%	–	–	–	–	3.3%	78	63	26.0	26.4
RCPSP/max	0.0%	–	–	–	–	0.0%	–	–	–	–	10.5%	3612	348	13.6	34.0

Table 7 – Continued

Dual fixings due to a single constraint (Lemma 6)															
TESTSETC	19.6%	234	14	2.2	2.1	19.6%	234	14	2.2	2.1	19.8%	236	14	2.2	2.1
TESTSETD	19.3%	236	15	2.3	2.2	19.3%	236	15	2.3	2.2	19.8%	243	15	2.3	2.2
J10	5.6%	16	2	1.1	0.2	5.6%	16	2	1.1	0.2	6.3%	19	2	1.1	0.3
J20	2.6%	7	1	1.0	0.0	2.6%	7	1	1.0	0.0	3.7%	10	1	1.0	0.0
J30	3.0%	9	2	1.1	0.3	3.0%	9	2	1.1	0.3	4.1%	13	2	1.2	0.4
UBO100	0.0%	–	–	–	–	0.0%	–	–	–	–	2.2%	2	1	1.0	0.0
UBO20	1.1%	1	1	1.0	0.0	1.1%	1	1	1.0	0.0	2.2%	3	2	1.5	0.5
UBO50	0.0%	–	–	–	–	0.0%	–	–	–	–	1.1%	1	1	1.0	0.0
RCPSP/max	9.6%	503	15	2.1	2.0	9.6%	503	15	2.1	2.0	10.2%	527	15	2.1	2.0
Dual fixings due to a set of constraints (Corollary 2)															
TESTSETC	44.4%	1555	45	6.5	7.8	44.6%	1556	45	6.5	7.8	44.6%	1561	46	6.5	7.9
TESTSETD	35.2%	1100	45	5.8	7.6	35.2%	1100	45	5.8	7.6	35.9%	1129	45	5.8	7.6
J10	5.2%	20	3	1.4	0.6	5.2%	20	3	1.4	0.6	6.7%	27	3	1.5	0.7
J20	3.7%	11	2	1.1	0.3	3.7%	11	2	1.1	0.3	5.9%	18	2	1.1	0.3
J30	3.7%	10	1	1.0	0.0	3.7%	10	1	1.0	0.0	4.1%	12	2	1.1	0.3
UBO10	8.9%	10	3	1.2	0.7	8.9%	10	3	1.2	0.7	13.3%	16	3	1.3	0.7
UBO100	1.1%	1	1	1.0	0.0	1.1%	1	1	1.0	0.0	7.8%	10	3	1.4	0.7
UBO20	8.9%	11	2	1.4	0.5	8.9%	11	2	1.4	0.5	10.0%	17	4	1.9	1.0
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
UBO50	1.1%	1	1	1.0	0.0	1.1%	1	1	1.0	0.0	3.3%	6	3	2.0	0.8
RCPSP/max	19.2%	2721	45	5.6	7.5	19.2%	2722	45	5.6	7.5	20.4%	2806	46	5.4	7.3
All dual reductions															
TESTSETC	46.7%	6933	0	27.5	35.8	46.9%	6936	0	27.4	35.8	52.0%	31762	0	113.0	382.7
TESTSETD	39.8%	4750	0	22.1	33.0	39.8%	4750	0	22.1	33.0	46.5%	57470	0	229.0	863.8
J10	9.6%	110	0	4.2	4.0	9.6%	110	0	4.2	4.0	38.1%	3142	0	30.5	40.7
J20	5.2%	51	0	3.6	2.0	5.2%	51	0	3.6	2.0	30.4%	6466	0	78.9	97.8
J30	5.6%	56	0	3.7	2.2	5.6%	56	0	3.7	2.2	25.9%	8935	0	127.6	182.2
UBO10	8.9%	49	0	6.1	3.5	8.9%	49	0	6.1	3.5	53.3%	1925	0	40.1	31.2
UBO100	1.1%	3	0	3.0	0.0	1.1%	3	0	3.0	0.0	22.2%	13847	0	692.4	801.7
UBO20	8.9%	57	0	7.1	2.5	8.9%	57	0	7.1	2.5	40.0%	3422	0	95.1	109.3
UBO200	2.2%	9	0	4.5	1.5	2.2%	9	0	4.5	1.5	12.2%	13463	0	1223.9	1199.9
UBO50	1.1%	4	0	4.0	0.0	1.1%	4	0	4.0	0.0	28.9%	6363	0	244.7	290.3
UBO500	0.0%	–	–	–	–	0.0%	–	–	–	–	4.4%	25083	0	6270.8	10213.0
RCPSP/max	21.5%	12022	225	22.2	33.0	21.5%	12025	225	22.1	32.9	37.0%	171878	23954	184.4	956.4