Interdisciplinary Public Policy Mainz

IPP

Gutenberg School of Management and Economics

& Research Unit "Interdisciplinary Public Policy"

Discussion Paper Series

Dual Inequalities for Stabilized Column Generation Revisited

Timo Gschwind, Stefan Irnich

July 2014

Discussion paper number 1407

Johannes Gutenberg University Mainz
Gutenberg School of Management and Economics
Jakob-Welder-Weg 9
55128 Mainz
Germany
wiwi.uni-mainz.de

Contact details

Timo Gschwind
Chair of Logistics Management
Gutenberg School of Management and Economics
Johannes Gutenberg Universität Mainz
Jakob-Welder-Weg 9
55128 Mainz
Germany

gschwind@uni-mainz.de

Stefan Irnich
Chair of Logistics Management
Gutenberg School of Management and Economics
Johannes Gutenberg Universität Mainz
Jakob-Welder-Weg 9
55128 Mainz
Germany

irnich@uni-mainz.de

Dual Inequalities for Stabilized Column Generation Revisited

Timo Gschwind[a], Stefan Irnich[a]

[a] *Chair of Logistics Management, Gutenberg School of Management and Economics, Johannes Gutenberg University Mainz, Jakob-Welder-Weg 9, D-55128 Mainz, Germany.*

Abstract

Column generation (CG) models have several advantages over compact formulations, namely, they provide better LP bounds, may eliminate symmetry, and can hide non-linearities in their subproblems. However, users also encounter drawbacks in the form of slow convergency a.k.a. the tailing-off effect and the oscillation of the dual variables. Among different alternatives for stabilizing the CG process, Ben Amor *et al.* [Ben Amor, H., Desrosiers, J., and Valério de Carvalho, J. M. (2006). Dual-optimal inequalities for stabilized column generation. Operations Research, 54(3), 454–463] suggest the use of dual-optimal inequalities (DOIs) in the context of cutting stock and bin packing problems. We generalize their results, provide new classes of (deep) DOIs, and show the applicability to other problems (vector packing, vertex coloring, bin packing with conflicts). We also suggest the dynamic addition of violated dual inequalities in a cutting-plane fashion and the use of dual inequalities that are not necessarily (deep) DOIs. In the latter case, a recovery procedure is needed to restore primal feasibility. Computational results proving the usefulness of the methods are presented.

Key words: Column generation, dual inequalities, stabilization

1. Introduction

Column generation (CG) has been proven a versatile tool for solving large-scale linear programs (LPs) with often more variables than explicitly representable by considering only a selected subset of them at a time. Such huge models may arise naturally or may result from a reformulation of an integer compact model using a Dantzig-Wolfe decomposition (Dantzig and Wolfe, 1960), which then leads to an integer CG approach also known as branch-and-price (Barnhart *et al.*, 1998; Lübbecke and Desrosiers, 2005). The resulting so-called extensive formulation has an enormous number of variables corresponding to extreme points and extreme rays of the domain chosen as the subproblem in the decomposition. While the huge number of variables in the extensive formulation let the model seem unattractive, it also contributes with some profound advantages: The extensive formulation has a tighter LP-bound if the subproblem does not posses the integrality property. Moreover, if non-linearities are present in the compact formulation, they may be hidden using the right definition of the subproblems. Additionally, some very successful CG-based algorithms have been invented for applications in, e.g., vehicle and crew routing and scheduling (Desaulniers *et al.*, 1998), cutting and packing (Ben Amor and Valério de Carvalho, 2005), and graph coloring (Held *et al.*, 2012). Their success can be attributed to the availability of very powerful subproblem solution algorithms that have reached a high degree of maturity, see (Irnich and Desaulniers, 2005) for constrained shortest path problems, (Kellerer *et al.*, 2004) for knapsack problems, and (Östergård, 2002; Carraghan and Pardalos, 1990) for cliques/stable sets.

However, CG approaches in practice often suffer from instability problems. The dual variables, which drive the generation process, may oscillate heavily before they finally converge to some optimal dual values. On the primal side, CG formulations often tend to be degenerated: In many applications the master program

Email addresses: gschwind@uni-mainz.de (Timo Gschwind), irnich@uni-mainz.de (Stefan Irnich)

is some (generalized) set-partitioning or set-covering problem, in which each variable represents a crew schedule, vehicle route, packing, or partition, which is often a dense column so that a few columns comprise a solution. However, a primal basis must include several other variables that are then at value zero. As a consequence, primal degeneracy leads to rather small and often non-improving LP pivots, known as the tailing-off effect (Gilmore and Gomory, 1961; Vanderbeck, 2005). The process of variable generation then continues over many iterations to produce nearly no improvement in the LP objective. In order to explicitly stabilize the dual values, algorithmic techniques like the box step method (Marsten *et al.*, 1975), bundle methods (Hiriart-Urruty and Lemaréchal, 1993), and tailored CG stabilization approaches have been proposed, e.g., by du Merle *et al.* (1999); Rousseau *et al.* (2007); Lee and Park (2011). Furthermore, a branch of the recent literature is especially devoted to tools that can help overcome or even benefit from primal degeneracy when solving huge LPs (Gauthier *et al.*, 2014; Desrosiers *et al.*, 2014).

In the paper at hand, we continue the path originally followed by Valério de Carvalho (2005) and Ben Amor *et al.* (2006). For the *cutting stock problem* (CS) and the *bin packing problem* (BP), they have shown that the knowledge of the domain of optimal dual values can be used for stabilizing the CG process. In particular, any valid inequality known for the optimal dual space can be added as an additional variable to the corresponding primal CG formulation.

We introduce the concept of dual inequalities in a more formal way now following the notation of Ben Amor *et al.* (2006): Consider a pair of primal and dual LPs of the forms

$$(P)\quad \begin{aligned} z_P = \min\ & c^\top \lambda \\ \text{s.t.}\ & A\lambda = b \\ & \lambda \geq 0 \end{aligned} \qquad\qquad (D)\quad \begin{aligned} z_D = \max\ & b^\top \pi \\ \text{s.t.}\ & A^\top \pi \leq c \end{aligned}$$

with a matrix $A = (a_{ij}) \in \mathbb{R}^{I \times J}$, $c = (c_i) \in \mathbb{R}^I_{>0}$, and $b = (b_i) \in \mathbb{R}^I_{>0}$ with row indices $i \in I$ and column indices $j \in J$. Let $m = |I|$ denote the number of rows. We refer to the jth column of A as $a_j \in \mathbb{R}^I$ such that $A = (a_j)_{j \in J}$. For any vector $a \in \mathbb{R}^I$, we write $a \in A$ if and only if $a = a_j$ for some $j \in J$. Rows of A are denoted by $a_{i*} \in \mathbb{R}^J$ for $i \in I$.

In the following, we always assume that the primal formulation P is the extensive formulation to which a CG algorithm is applied. Therefore, we do not solve P directly, but work with a restricted version of P, the *restricted master program* (RMP), for which re-optimization of the LP and the solution of the pricing problem (=subproblem) alternate. The pricing problem asks for the determination of at least one column $a_j \in A$, for which $c_j - \pi^\top a_j < 0$ holds, or the guarantee that no such column exists. In the latter case, P has been solved to optimality and the CG process terminates.

The idea of Ben Amor *et al.* (2006) was to add additional constraints $E^\top \pi \leq e$ to the dual D. Additional constraints in the dual D correspond with additional variables in the primal P, denoted by y in the following. The extended primal and dual models are:

$$(\tilde{P})\quad \begin{aligned} z_{\tilde{P}} = \min\ & c^\top \lambda + e^\top y \\ \text{s.t.}\ & A\lambda + Ey = b \\ & \lambda \geq 0, y \geq 0 \end{aligned} \qquad\qquad (\tilde{D})\quad \begin{aligned} z_{\tilde{D}} = \max\ & b^\top \pi \\ \text{s.t.}\ & A^\top \pi \leq c \\ & E^\top \pi \leq e. \end{aligned}$$

The set of additional *dual inequalities* (DIs) $E^\top \pi \leq e$ cuts part of the dual solution space. Thus, $\tilde{D}$ is a restriction of D, whereas the corresponding primal model $\tilde{P}$ is a relaxation of P. We denote by D^* the set of optimal solutions to the models D, i.e., the dual-optimal space. Throughout the text, we heavily use acronyms; for convenience, Table 1 lists those that are most frequently used.

One can further differentiate two special classes of DIs depending on which part of the dual solution space they cut off (see Ben Amor *et al.*, 2006, p. 455): DIs $E^\top \pi \leq e$ are called *dual-optimal inequalities* (DOIs) if *all* dual-optimal solutions $\pi^* \in D^*$ also satisfy the DIs, i.e., $D^* \subseteq \{\pi : E^\top \pi \leq e\}$. If DIs $E^\top \pi \leq e$ are satisfied by at least one dual-optimal solution $\pi^* \in D^*$, i.e., $D^* \cap \{E^\top \pi \leq e\} \neq \emptyset$, they are called a set of *deep dual-optimal inequalities* (DDOIs). Note that deep dual-optimal is a property referring to a set of DIs, meaning that DDOIs cannot be tested individually for each inequality in $E^\top \pi \leq e$. In contrast, for a

Acronym	Full Name
CG	column generation
RMP	restricted master program
DI	dual inequality; any inequality in the variables of D
DOI	dual-optimal inequality; fulfilled by every dual-optimal solution of D
DDOI	(set of) deep dual-optimal inequality/ies
WSI	weighted subset inequality
SI	subset inequality
CS	cutting stock problem
BP	bin packing problem
BPC	bin packing problem with conflicts
VP	vector packing problem
VC	vertex coloring problem
KP	binary knapsack problem
UKP	unbounded knapsack problem
KPC	binary knapsack problem with conflicts
DKP	unbounded version of the d-dimensional knapsack problem
MWIS	maximum weight independent set

Table 1: Some acronyms used in the paper

set of DOIs, every single DI $E_{j*}\pi \leq e_j$, where E_{j*} refers to the jth row of E, must qualify as a DOI. This is fulfilled if all dual-optimal solutions $\pi^* \in D^*$ respect $E_{j*}\pi^* \leq e_j$.

The contribution of our paper consists in at least five new findings: First, we generalize Propositions 1 and 2 of Ben Amor *et al.* (2006) and Proposition 1 of Valério de Carvalho (2005) providing insights into the relations of optimal solution values and optimal solutions to models P, D, $\tilde{P}$, and $\tilde{D}$ when $E^\top \pi \leq e$ are DOIs or DDOIs. Second, we define several new classes of properties for integer valued matrices A and unit costs $c = \mathbf{1}$ that enable the identification of new DOIs and DDOIs. Herewith, we derive additional classes of DDOIs for BP and DOIs for CS. The latter can be extended also to the *vector packing problem* (VP) which can be seen as the multi-dimensional generalization of CS.

The third aspect is the extension of the DI-based stabilization to several new classes of problems: We address the *vertex coloring problem* (VC) in which the task is to color the vertices of a given undirected graph with a minimum number of colors so that adjacent vertices receive different colors. The synthesis of VC and BP is the *bin packing problem with conflicts* (BPC). In all cases the derivation of DOIs and DDOIs is based on showing that the above mentioned new matrix properties are valid.

Fourth, we show that a kind of over-stabilization of CG procedures can result in an overall faster convergency. There exist cases in which one can suspect that $E^\top \pi \leq e$ are DDOIs, but this may actually not be true. The property may be fulfilled just for some instances of the problem or just for a proper subset that one is unable to identify. In the negative case, the outcome of the stabilized CG algorithm is an infeasible primal solution and a weaker bound than z_P (see Proposition 1). We show that there is a constructive way of identifying the DIs that have cut off D^*. This can be seen as a *recovery procedure*. In order to accelerate the overall CG approach, we therefore propose to alternate between solving the (over-)stabilized formulation $\tilde{P}$ and the recovery procedure until a feasible primal solution results.

Fifth and finally, we show that the dynamic generation of DIs during the CG process is another option that often helps reducing the computation time. Note that up to now, DOIs and DDOIs have been used in a static fashion. Indeed, when the number of known DOIs or DDOIs is relatively small, e.g., polynomial in the size of the input instance, then $\tilde{P}$ is not significantly larger than P, and the advantage of stability can then outreach the resulting larger RMP re-optimization times. Such a family of DOIs has been used in a branch-and-price approach for the capacitated arc-routing problem (Bode and Irnich, 2012). However, in all problems considered here, the families of DIs, DOIs, and DDOIs are exponential in size. For the problems CS, BP, VC, and BPC we demonstrate that often the dynamic generation, sometimes in combination with an a priori addition of the expectedly strongest DIs provides the best trade-off between stabilization, effort of DI generation, and effort for re-optimizing the RMP.

The remainder of this paper is organized as follows: Section 2 presents the generalized proposition on the relations of solutions of models P, D, $\tilde{P}$, and $\tilde{D}$. In Section 3, we derive useful properties of the coefficient matrix allowing to identify DIs that are DOIs and/or DDOIs. Along with introducing the new properties we present their application to different problems. This includes the proper introduction of the CG formulations of these problems and the specification of the different new classes of DOIs and DDOIs. Section 4 clarifies the dynamic identification of violated DIs and presents the recovery procedure. In Section 5, we present aggregated computational results for BP, CS, VC, and BPC. Final conclusions are drawn in Section 6.

2. Equivalence Conditions for the Original and Extended Models

The central question of this section is the following: Under which conditions are the models P and $\tilde{P}$ and the models D and $\tilde{D}$ equivalent? Conditions for the equivalence depend on the set of DIs $E^\top\pi \leq e$. Equivalence of the respective models means that they provide identical LP-bounds. Moreover, it is required that optimal solutions to the extended models $\tilde{P}$ and $\tilde{D}$ can be transformed into optimal solutions of the corresponding original models P and D, respectively. In case of equivalence, the stabilized model $\tilde{P}$ can now be solved by CG instead of the non-stabilized model P.

The next proposition generalizes the findings of Ben Amor *et al.* (2006) also including some results of Valério de Carvalho (2005). In essence, if the $E^\top\pi \leq e$ are DDOIs then the models $P, D, \tilde{P}$, and $\tilde{D}$ are equivalent, and vice versa.

Proposition 1. *The following statements are equivalent:*

(i) $E^\top\pi \leq e$ *are DDOIs.*
(ii) *There exists a* $\pi^* \in D^*$ *which is feasible also for* $\tilde{D}$.
(iii) $z_D = z_{\tilde{D}}$.
(iv) $z_P = z_{\tilde{P}}$.
(v) *For every feasible primal solution* $(\tilde{\lambda}, y)$ *to* $\tilde{P}$ *there exists a primal feasible solution* λ *to* P *with* $c^\top\lambda \leq c^\top\tilde{\lambda} + e^\top y$.
(vi) *There exists a primal optimal solution* $(\tilde{\lambda}^*, y^*)$ *to* $\tilde{P}$ *with* $Ey^* = 0$. *Also,* $\tilde{\lambda}^*$ *is an optimal solution to* P.
(vii) *Every optimal dual solution* π^* *to* $\tilde{D}$ *is optimal for* D.

The proof of Proposition 1 and proofs for all other propositions and theorems can be found in Section A of the Appendix.

Ben Amor *et al.* (2006) have shown the implications (i) $\Rightarrow$ (iii), (i) $\Rightarrow$ (iv), and (i) $\Rightarrow$ (vii). Moreover, for DOIs they showed that if there exists a primal optimal solution $(\tilde{\lambda}^*, y^*)$ to $\tilde{P}$ with $Ey^* = 0$, then $\tilde{\lambda}^*$ is optimal for P. Since DOIs are also DDOIs, Proposition 1 proves the fact that if $E^\top\pi \leq e$ are DOIs then this implies (i)–(vii). The reverse does not hold in general.

From a practical point of view, the condition (v) is particularly useful for problems for which a constructive procedure is known to transform solutions $(\tilde{\lambda}, y)$ of $\tilde{P}$ so that they lead to a solution λ of P with less or equal cost. A solution to model P can then be determined by first solving the relaxed model $\tilde{P}$ to optimality, and then converting this solution into a solution to P. This approach has been taken by Valério de Carvalho (2005) when solving CS with CG: He first solves the extended master program $\tilde{P}$, in which several additional y variables corresponding to DIs have been added a priori. As a result, the optimal master program may contain positive y variables. This solution is then transformed into a feasible CS solution with identical cost (our recovery procedure presented in Section 4.2 is inspired by this transformation). Thus, Valério de Carvalho (2005) was the first to show that the original CS problem is actually solved to optimality, even if additional y variables are active. In fact, the DIs that he used for CS are DOIs as later shown by Ben Amor *et al.* (2006).

Even if there is no direct transformation from solutions $(\tilde{\lambda}, y)$ of $\tilde{P}$ into solutions λ of P known, the model P can still be solved by solving a stabilized variant of the model $\tilde{P}$. Ben Amor *et al.* (2006) show that for DOIs it suffices to marginally increase the costs e of the y variables to $e + \varepsilon$. As a result, all DOIs are inactive and the corresponding primal variables y must be at 0 (resulting from complementary

slackness). Moreover, Ben Amor *et al.* (2006) suggest a two-phase procedure for DDOIs: In the first phase, they determine an optimal solution to $\tilde{P}$. It has a corresponding dual solution π^*. In the second phase, they use a stabilized model P, in which the dual variables π are stabilized with the help of a trust region around π^*, and deviations are penalized. The primal solution λ^* to this stabilized model is then a feasible solution to P, see (Ben Amor *et al.*, 2006, Prop. 5).

3. Matrix Properties for Deriving DOIs and DDOIs

The definition of DOIs and DDOIs refers to the set of dual-optimal solutions D^*. Up to now, DOIs and DDOIs have been derived by analyzing structural properties of the problem at hand, i.e., for CS and BP. In this section, we derive some new results by analyzing the structure of the constraint matrix A defining the model P. This allows us to apply the results to new and different problems showing that several classes of DIs are DOIs or DDOIs.

Some additional notation is needed: Recall that I are the row indices and J are the column indices of the coefficient matrix A. We denote the ith unit vector for $i \in I$ by $u_i \in \mathbb{Z}_+^I$. Moreover, $\mathbf{0}$ and $\mathbf{1}$ are vectors with all entries 0 and 1, respectively, of appropriate size. In the following, we consider an integer valued matrix $A \in \mathbb{Z}_+^{I \times J}$, an integer valued, strictly positive right-hand side (rhs) $b \in \mathbb{Z}_{>0}^I$ and unit costs $c = \mathbf{1} \in \mathbb{R}^J$.

3.1. Exchange Property

Several results presented in the following use the following exchange property:

Definition 1. (Exchange Property) Let $s, t \in \mathbb{Z}_+^I$ be given. A matrix $A \in \mathbb{Z}_+^{I \times J}$ has the (s,t)-*exchange property* if $a_j \in A, a_j \geq s$ implies $a_j - s + t \in A$.

If one or both vectors s and t are unit vectors, e.g., $s = u_h$, we simplify the notation and write (h,t)-exchange property. If one of the vectors is the null vector, e.g., $t = \mathbf{0}$, we write $(s,0)$-exchange property.

The exchange property means that the columns in A are related to each other. Any column that is not too small, i.e., fulfills $a_j \geq s$, can be converted into another column by exchanging entries in some rows against entries in other rows (described by $t - s$). In this case, useful relations between optimal values of primal and dual variables can be derived:

Proposition 2. *Let $s = (s_i), t = (t_i) \in \mathbb{Z}_+^I$ and $A \in \mathbb{Z}_+^{I \times J}$ be given. If A has the (s,t)-exchange property and (λ^*, π^*) is a pair of optimal solutions to P and D, then*

$$\text{(Exch-D)} \quad \sum_{i \in I} s_i \pi_i^* \geq \sum_{i \in I} t_i \pi_i^* \qquad \text{or} \qquad \text{(Exch-P)} \quad \lambda_k^* = 0 \ \text{ for all } k \in J \text{ with } a_k \geq s.$$

If s is a unit vector u_h for $h \in I$, the exchange property allows that an entry in row h is replaced by entries in a subset of other rows given by the (strictly) positive entries in t. The following proposition shows that valid DOIs result for the dual variables described by h and the positive components of t.

Proposition 3. *Let $h \in I$ and $t \in \mathbb{Z}_+^I$ be given. If a matrix $A \in \mathbb{Z}_+^{I \times J}$ has the (h,t)-exchange property, then any dual-optimal solution $\pi^* \in D^*$ fulfills*

$$\pi_h^* \geq \sum_{i \in I} t_i \pi_i^*.$$

Proposition 3 can directly be applied to identify DOIs for problems whose coefficient matrix A has the (h,t)-exchange property. We exemplify this for CS and VP.

Cutting Stock. The CS consists of finding cutting patterns for cutting rolls of length L into items $i \in I$ of length $w_i \leq L$ such that the total number of rolls is minimal and given demands b_i of the items $i \in I$ are fulfilled. A feasible cutting pattern cuts a roll into several of the items with $\sum_{i \in I} a_{ij} w_i \leq L$, where a_{ij} denotes the number of items of length w_i that are produced by pattern $j \in J$. An extensive formulation for CS was first presented by Gilmore and Gomory (1961), and several CG-based algorithms have been presented over the years, e.g., by Valerió de Carvalho (1998); Vanderbeck (2000); Ben Amor and Valério de Carvalho (2005). In out notation, model P for CS consists of columns $a_j \in \mathbb{Z}_+^I$ forming A, one for each feasible cutting pattern. The variables λ_j give the number of rolls that are cut according to pattern j (cf. Gilmore and Gomory, 1963). The pricing problem has to identify a cutting pattern with negative reduced cost. This is an *unbounded knapsack problem* (UKP), which can be solved by dynamic programming algorithms as a *longest path problem* in an acyclic digraph (see Kellerer *et al.*, 2004). We provide some more details in Section 4.1.

Vector Packing. The VP is the d-dimensional ($d \geq 2$) generalization of CS. CG algorithms for VP have been presented by Caprara and Toth (2001) and Alves *et al.* (2014). In VP, items $i \in I$ have a size $w_{i1}, w_{i2}, \ldots, w_{id}$ in each of the d dimensions (e.g., weight, volume, cost etc.), and *cutting patterns* or *packings* are restricted not only by a single capacity, but by d independent capacities $L_1, L_2, \ldots, L_d$. A straightforward CG model for VP is analog to the one for CS: Columns in the master program correspond to feasible packings, while the subproblem is the *unbounded* version of a *d-dimensional knapsack problem* (DKP) (see Kellerer *et al.*, 2004, Ch. 9).

A direct consequence of Proposition 3 is:

Theorem 1. *Let $d = 1$ or $d \geq 2$. Moreover, let $w_i, i \in I$ (for $d = 1$) or $w_{ip}, i \in I, p \in \{1, 2, \ldots, d\}$ (for $d \geq 2$) be the d-dimensional weights defining CS and VP, and let $h \in I$, $S \subseteq I \setminus \{h\}$, and $t \in \mathbb{Z}_{>0}^S$ be given. We distinguish between $d = 1$ and $d \geq 2$:*

(i) *If $w_h \geq \sum_{s \in S} t_s w_s$, then the inequality $\pi_h \geq \sum_{s \in S} t_s \pi_s$ is a DOI for CS.*
(ii) *If $w_{hp} \geq \sum_{s \in S} t_s w_{sp}$ for all $p = 1, \ldots, d$, then the inequality $\pi_h \geq \sum_{s \in S} t_s \pi_s$ is a DOI for VP.*

In the following, we refer to DIs of the form $\pi_h \geq \sum_{s \in S} t_s \pi_s$ as *weighted subset inequalities* (WSIs). In the primal model $\tilde{P}$, the corresponding kth column to a WSI is $(E_{ik})_{i \in I} \in \mathbb{Z}^I$ with

$$(E_{ik}) = \begin{cases} t_i & i \in S, \\ -1 & i = h, \\ 0 & otherwise, \end{cases}$$

and a cost of zero, i.e., $e_k = 0$. We refer to such a WSI and its (so-called) WSI column as $(h \leftarrow t, S)$ with $h \in I$, $S \subseteq I \setminus \{h\}$, and $t \in \mathbb{Z}_{>0}^S$ instead of using the column index k. Moreover, for $t = \mathbf{1} \in \mathbb{Z}_{>0}^S$, the DI $\pi_h \geq \sum_{s \in S} \pi_s$ is a *subset inequality* (SI), and we refer to it and its column as $(h \leftarrow S)$.

The WSIs and the associated primal WSI columns have a very intuitive practical interpretation in CS: The quantities t_s for $s \in S$ describe a combination of items including copies whenever $t_s > 1$. Whenever the total length of the combination is not longer than the length w_h of item h, then it is more difficult to include item h in a cutting pattern than all items (and their copies) of the combination. Hence, the marginal cost π_h of producing item h should be not smaller than the overall marginal cost of producing the combination. It also means that in any cutting pattern that includes item h one can safely replace this item h by the combination, i.e., all items $s \in S$ in quantities given by $t_s, s \in S$. The result is a different, but certainly feasible cutting pattern.

When $\tilde{P}$ is solved by CG, the presence of a WSI $(h \leftarrow t, S)$ in the RMP implicitly represents several other cutting pattern columns. For any cutting pattern column a_j including item h, i.e., $a_j \geq u_h$, the sum of column a_j and a WSI column $(h \leftarrow t, S)$ realizes the cutting pattern in which items $s \in S$ are cut an additional t_s times and item h is cut once fewer than given by a_{hj}. Thus, the presence of WSI columns in the RMP prevents having to explicitly generate specific columns because they are already implicitly represented.

Moreover, these implicitly represented cutting patterns and other WSIs together represent additional cutting pattern columns.

Note that the WSIs for CS in Theorem 1 are generalizations of the SIs introduced by Valério de Carvalho (2005) and proven to be DOIs by Ben Amor *et al.* (2006). Both works exclusively consider DOIs of the form $\pi_h \geq \sum_{s \in S} \pi_s$, hence they neglect the possibility to replace a single item by multiples of one or more items $s \in S$.

3.2. Covering Property

It seems to be common knowledge or folklore that in CG (generalized) covering formulations are preferable over partitioning formulations, i.e., inequality over equality constraints. In the light of Proposition 3 we can justify this as follows: If the matrix A has the $(h, 0)$-exchange property, it means that matrix entries in the row h can be decreased as long as a column $a_j, j \in J$ has a positive entry a_{hj}. Intuitively, the hth equality of $A\lambda = b$ can then be replaced by a covering constraint. The following remark formalizes the idea from a primal and dual perspective:

Remark 1. Let a matrix $A \in \mathbb{Z}_+^{I \times J}$ have the $(h, 0)$-exchange property for every $h \in I$. Then:

(i) For every primal solution λ to the system $A\lambda \geq b, \lambda \geq 0$ there exists another primal solution λ' to P with $\mathbf{1}^\top \lambda' = \mathbf{1}^\top \lambda$.

(ii) Any dual-optimal solution $\pi^* \in D^*$ fulfills $\pi_h^* \geq 0$.

The above result can be used to explain why some (generalized) partitioning problems are equivalent to their covering versions, which is a well-known fact for many problems. A prerequisite is that the coefficient matrix A of the problem has the $(h, 0)$-exchange property for every $h \in I$. Practically speaking, a subset of a feasible structure is again a feasible structure of the same type. This is known as the concept of *hereditary* (see, e.g., Pattillo *et al.*, 2013). For CS, e.g., removing an item from a cutting pattern clearly results in another feasible cutting pattern.

Note that for solving these equivalent formulations by CG the covering formulation (with $A\lambda \geq b$) is significantly more stable than the one with equality ($A\lambda = b$). In the latter, the feasible dual space is in $\mathbb{R}^I$, while feasible dual solutions for covering come from the positive quadrant $\mathbb{R}_+^I$. Hence, the dual space is reduced by a factor of 2^m.

Hereditary problems are VP and BP. Additional examples are VC, which is equivalent to partitioning with independent sets and partitioning with cliques in the complement graph, the *edge coloring problem*, and partitioning with matchings. In the context of vehicle routing, most tour minimization problems are hereditary. In all these problems (some are considered and formally introduced later in this article), subsets of packings, independent sets, cliques, and matchings are clearly feasible subsets and, hence, partitioning and covering are equivalent formulations.

For other problems, however, covering and partitioning differ: Covering and partitioning a graph with certain types of subgraphs are different problems if the subgraphs are, e.g., cycles or k-clubs. Here the subgraph formed by a subset of the vertices of a cycle or a k-club is generally not a cycle or k-club (see Pattillo *et al.*, 2013). The same is true for routing problems with a maximal waiting time constraint, since a subtour of a feasible tour may violate the waiting time constraint.

3.3. Row Interchange Properties

We now consider linear programs with only binary coefficients so that $A \in \mathbb{B}^{I \times J}$ and $b = \mathbf{1}$ holds. For these we define a modified version of the exchange property:

Definition 2. (Row Replacement Property) Let $h, i \in I$ be given. A binary matrix $A \in \mathbb{B}^{I \times J}$ has the (h, i)*-row replacement property* if $a_j \in A, a_j \geq u_h$ and $a_{ij} = 0$ implies $a_j - u_h + u_i \in A$. In this case, the pair $(h, i) \in I \times I$ is called a *valid replacement* for A.

The analog to Proposition 3 for linear problems P whose coefficient matrix A satisfies the row replacement property is:

Proposition 4. *Let $E^\top \pi \leq e$ be the set of all inequalities $\pi_i - \pi_h \leq 0$ for valid replacements (h, i) for $A \in \mathbb{B}^{I \times J}$. Then, $E^\top \pi \leq e$ are DDOIs.*

Using Proposition 4, we can show that specific classes of DIs for BP, VC, and BPC are DDOIs. We briefly introduce these problems now.

Bin Packing. The BP is a restricted CS in which all items $i \in I$ have unit demand $b_i = 1$. Consequently, an item cannot appear more than once in a feasible cutting pattern, called *bin* in BP. Thus, A is a binary matrix ($A \in \mathbb{B}^{I \times J}$) and the CG pricing problem is a *binary knapsack problem* (KP) (see Kellerer *et al.*, 2004). CG algorithms for BP have been suggested by Gilmore and Gomory (1963); Vance *et al.* (1994); Valério de Carvalho (1999). Clearly, for BP, (h, i) is a valid replacement for A if $w_h \geq w_i$.

Vertex Coloring. The VC is defined for an undirected graph $G = (I, \mathcal{E})$ with vertex set I and edge set $\mathcal{E}$. VC is the problem of feasibly coloring the vertices with a minimum number of colors such that any two adjacent vertices receive different colors. We denote by $N(i)$ the set of vertices adjacent to vertex $i \in I$. An *independent set* in G is a subset $S \subseteq I$ such that no two vertices of S are adjacent. Clearly, in VC all vertices of the same color form an independent set. CG models for VC were analyzed, e.g., in (Mehrotra and Trick, 1996; Malaguti *et al.*, 2011; Gualandi and Malucelli, 2012; Held *et al.*, 2012), and they are defined as follows: The columns $a_j \in \mathbb{B}^I$ of A are incidence vectors of independent sets, and the task is to properly partition I into independent sets, i.e., the rhs is $b = \mathbf{1}$. The pricing problem in this formulation consist of finding a *maximum-weight independent set* (MWIS).

Bin Packing with Conflicts. The BPC is the synthesis of BP and VC: Items $i \in I$ with unit demand $b_i = 1$ and weights w_i have to be packed into a minimum number of bins each with a capacity of L. Moreover a conflict graph $G = (I, \mathcal{E})$ with vertex set I and edge set $\mathcal{E}$ is given, where an edge $\{i, j\}$ indicates that the items i and j are in conflict. Conflicting items cannot be packed into the same bin. A CG formulation for BPC is analog to the formulations for BP or VC and has been presented by Sadykov and Vanderbeck (2013). Columns can equivalently be seen as *conflict free bins* or *capacity constrained independent sets*.

We refer to a DI of the form $\pi_h \geq \pi_i$ as *pair inequality*. Clearly, the pair inequalities are a special case of the SIs.

Theorem 2.

(a) *The pair inequalities $\{\pi_h \geq \pi_i : w_h \geq w_i\}$ are DDOIs for BP.*
The pair inequality $\pi_h \geq \pi_i$ is a DOI for BP, if in addition $w_h + w_i > L$ holds.
The SI $\pi_h \geq \sum_{s \in S} \pi_s$ is a DOI for BP, if $w_h \geq \sum_{s \in S} w_s$ and $w_h + \min_{s \in S} w_s > L$.

(b) *The pair inequalities $\{\pi_h \geq \pi_i : N(h) \cup \{h\} \supseteq N(i)\}$ are DDOIs for VC.*
The pair inequality $\pi_h \geq \pi_i$ is a DOI for VC, if in addition $h \in N(i)$ holds.
The SI $\pi_h \geq \sum_{s \in S} \pi_s$ is a DOI for VC, if $N(h) \cup \{h\} \supseteq N(s)$ and $h \in N(s)$ for all $s \in S$, and S is an independent set.

(c) *The pair inequalities $\{\pi_h \geq \pi_i : w_h \geq w_i, N(h) \cup \{h\} \supseteq N(i)\}$ are DDOIs for BPC.*
The pair inequality $\pi_h \geq \pi_i$ is a DOI for BPC, if in addition $w_h + w_i > L$ or $h \in N(i)$ holds.
The SI $\pi_h \geq \sum_{s \in S} \pi_s$ is a DOI for BPC, if $w_h \geq \sum_{s_\in S} w_s$, $N(h) \cup \{h\} \supseteq N(s)$ for all $s \in S$, S is an independent set, and $w_h + \min_{s \in S} w_s > L$ or $h \in N(s), s \in S$ holds.

The above pair inequalities of Theorem 2(a) are DOIs for CS and VP, while they are only DDOIs for BP. Note that for BP the *equality constraints* $\pi_h = \pi_i$ for items $h, i \in I$ with $w_h = w_i$ used by Ben Amor *et al.* (2006) are a special case of the DDOIs of Theorem 2(a). Equality constraints are analyzed in a more general form in the next Section 3.4.

Also note that for BP and CS a linear-sized subset of the DIs of Theorem 2 is sufficient to enforce all $\mathcal{O}(m^2)$ pair inequalities: One can assume that all items are sorted by non-decreasing weights so that $w_1 \leq w_2 \leq \cdots \leq w_m$. Then, the $m-1$ *ranking inequalities* $\pi_1 \leq \pi_2 \leq \cdots \leq \pi_m$ imply all pair inequalities (cf. Valério de Carvalho, 2005; Ben Amor *et al.*, 2006, for CS). For VP, VC, and BPC such an ordering of the vertices/items is generally not possible, and no linear system of DIs can impose all pair inequalities.

3.4. Constraint Aggregation and Elimination

It is well known for BP that equally-sized items $i_1, \ldots, i_p \in I$ with $w_{i_1} = \cdots = w_{i_p}$ can be aggregated (Vanderbeck, 1999; Ben Amor *et al.*, 2006). It means that the p rows $i_1, \ldots, i_p$ are dropped and a new row, say row $k \in I$, is introduced instead. The result is a master program, in which the aggregated row has rhs $b_k = p$, and the new entry of a column a_j is the sum $\sum_{\ell=1}^{p} a_{i_\ell,j}$ so that a_{kj} can take the values from $\{0, 1, \ldots, p\}$. Consequently, the pricing problem of an CG approach to BP with aggregation is a *bounded knapsack problem*, see (Kellerer *et al.*, 2004). Note that a similar aggregation is also possible for CS and VP.

In the context of DIs, Ben Amor *et al.* (2006) have characterized this principle of aggregation for BP in more detail. They have shown that the set of *equality constraints* $\pi_h = \pi_i$ for all $h, i \in I$ with $w_h = w_i$ is a set of DDOIs for BP. Furthermore, they proposed two different ways to enforce the equality constraints in a CG algorithm: explicitly adding the corresponding primal columns to the RMP or using constraint aggregation. Their computational results indicated that constraint aggregation is by far superior, which can be attributed to the following facts (see Ben Amor *et al.*, 2006): The size of the RMP decreases significantly in terms of both the number of constraints and the number of feasible packings. The size of the subproblem is also reduced.

With the following proposition we generalize the simple addition of two rows:

Proposition 5. *Let $\alpha \in \mathbb{R}$, $h, i \in I$ be two row indices with the primal constraints $a_{h*}\lambda = b_h$ and $a_{i*}\lambda = b_i$ and associated dual variables π_h and π_i, respectively.*

(i) *The following constraints are equivalent:*

$$(a_{h*} + \alpha a_{i*})\tilde{\lambda} = b_h + \alpha b_i \qquad \Leftrightarrow \qquad \begin{pmatrix} a_{h*} \\ a_{i*} \end{pmatrix} \tilde{\lambda} + \begin{pmatrix} \alpha \\ -1 \end{pmatrix} y = \begin{pmatrix} b_h \\ b_i \end{pmatrix}, \qquad y \in \mathbb{R} \tag{1}$$

(ii) *Let $\pi^* \in D^*$ be a dual-optimal solution with $\alpha\pi_h^* = \pi_i^*$, which fulfills $E^\top \pi^* \leq e$ for a set of given DDOIs. (This is equivalent to stating that $\alpha\pi_h = \pi_i$ together with $E^\top \pi \leq e$ are DDOIs.)*
Then, P and $\tilde{P}$ are equivalent to an aggregated formulation $\tilde{P}'$ in which the hth and ith equalities are replaced by the lhs of (1). The dual solution π'^* defined by

$$\pi_k'^* = \begin{cases} \pi_k^* & \text{for } k \in I \text{ with } k \neq h, i \\ \pi_h^* & \text{for the new aggregated constraint} \end{cases}$$

is an optimal solution to the dual of $\tilde{P}'$.

Proposition 5 gives rise to the following interesting result for instances of CS with no loss.

Remark 2. Consider an instance of CS which has a solution without loss. An optimal solution then uses the minimum number of rolls given by $\sum_{i \in I} w_i b_i / L$. Ben Amor *et al.* (2006) have shown that the DIs $\pi_i^* = w_i/L, i \in I$ are a set of DDOIs. Using this information about optimal dual values, the repeated aggregation of rows results in a corresponding LP with a single row, rhs $b' = \sum_{i \in I} b_i w_i / L$, and coefficients $a'_j \leq 1$ for all $j \in J$. Those aggregated columns with entry $a'_j = 1$ correspond to original columns $a_j, j \in J$ that represent a cutting pattern without loss. In the aggregated RMP, exactly one of these columns forms the basic solution. This variable takes the value $\sum_{i \in I} w_i b_i / L$.

The fact that the aggregated model can be solved by finding a cutting pattern without loss seems appealing. However, this result is not useful for computing a primal feasible LP-solution for the original model P. By disaggregation one can find a solution to $\tilde{P}$, where the above DDOIs are generally no WSIs. Thus, the recovery algorithm is not applicable (see Section 4.2). The only tool to transform the solution to $\tilde{P}$ with some active DDOIs is running a stabilized CG algorithm as discussed at the end of Section 2.

The above example of CS instances without loss is an extreme example in the sense that all constraints can be aggregated into a single constraint. In contrast, if only a partial aggregation but with $\alpha \neq 1$ is possible, one can benefit from Proposition 5 and the aggregated formulation because CG can then be applied to the corresponding non-trivial, but smaller instance. The result, the LP bound and the optimal dual values, both for the aggregated formulation and herewith also for the original formulation, may be rather helpful: Faster bounding procedures can be developed and optimal dual values can stabilize the CG algorithm for the original disaggregated model $\tilde{P}$, see again Section 2.

We now consider the special case of $\alpha = 0$ in Proposition 5. It means that the ith constraint is completely eliminated. The elimination of dominated constraints has been used long since in order to reduce the size of set-covering/set-partitioning instances (e.g., Balinski, 1965; Garfinkel and Nemhauser, 1969; Balas and Padberg, 1976): If for two rows $h, i \in I$ the inequality $a_{ij} \geq a_{hj}$ holds for all $j \in J$, then row i is dominated by row h and can be eliminated. Using this property, it is possible to eliminate specific rows in the CG formulation for VC:

Proposition 6. *Consider two vertices $h, i \in I$ in VC. If $N(h) \supseteq N(i)$, row i is dominated by row h and can therefore be eliminated.*

The positive effects of constraint elimination on a CG approach to VC are similar as those of aggregation for BP, CS, and VP. One can solve VC for the graph induced by the non-eliminated vertices, which leads to a row- and column-reduced RMP as well as smaller subproblem instances.

Following Proposition 5, the elimination of row i is one possibility to enforce the DI $\pi_i = 0$. Let P' be the primal model resulting from eliminating all dominated rows $I' \subseteq I$ from P. Because $z_{P'} = z_P$, the set of DIs $\pi_i = 0, i \in I'$ is a set of DDOIs (Proposition 1). An alternative way of enforcing the DI $\pi_i = 0$ is to explicitly include the corresponding column, i.e., the ith unit vector with cost zero, in the RMP. As for constraint aggregation and equality constraints, constraint elimination is computationally superior to an extended formulation with the DDOIs.

4. Separation and Recovery Algorithms

Our approach differs in several aspects from the approaches of Valério de Carvalho (2005) and Ben Amor *et al.* (2006) for stabilizing CG with DOIs or DDOIs. From an application point of view, we cover additional problems (VP, VC, and BPC) and provide general insights on how DOIs and DDOIs may be determined for alternative problems. The focus of this section is, however, on the differing algorithmic parts.

First, because the number of known DOIs and DDOIs is exponential in size, Valério de Carvalho (2005) and Ben Amor *et al.* (2006) suggested using only a linear-sized subset of the SIs with $|S| \leq 2$ when solving CS and BP. The associated primal variables y are here added as additional columns to $\tilde{P}$ before solving the RMP for the first time. We do not limit our approach in this way, but consider exponential classes of DOIs and DDOIs, which makes it necessary to identify violated DIs dynamically in the CG process. Indeed, this is a separation problem, and the first part of this section presents effective separation algorithms for CS, BP, VC, and BPC. The result of separation is the identification of a violated DI for $\tilde{D}$, which is then added as a new column to the primal formulation $\tilde{P}$. In this sense, separation of DIs is also column generation.

Second, we do not restrict our approaches to DIs that have been proven to be DOIs or DDOIs for the problem at hand. Instead, we may perform a kind of over-stabilization by also using classes of DIs that are generally neither DOIs nor DDOIs. As a result, we might end up with an optimal solution to the over-stabilized primal $\tilde{P}$ that cannot be transformed into a feasible solution to P (see Proposition 1). In these cases, we apply a recovery procedure to restore primal feasibility. Such a procedure is presented in the second part of this section.

In the following, all classes of DIs under consideration are in fact WSIs so that we present separation algorithms and the recovery algorithm for these. Recall that WSIs and SIs are denoted by $(h \leftarrow t, S)$ and $(h \leftarrow S)$, respectively. Clearly, one should only consider such subclasses of WSIs that are likely to be DDOIs at least for some instances. For pure binary problems such as BP, VC and BPC, we therefore restrict ourselves to SIs $(h \leftarrow S)$.

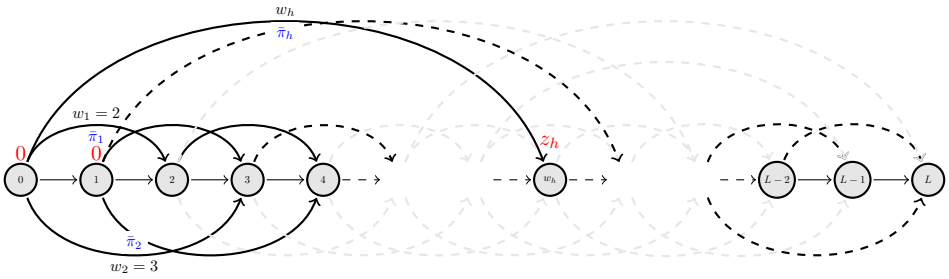

Figure 1: Dynamic programming state graph for the unbounded knapsack problem (UKP)

A WSI $(h \leftarrow t, S)$ or SI $(h \leftarrow S)$ is probably no DDOI if it violates the weight inequality for CS and BP, if the defining set S is no independent set for VC, and both for BPC. Therefore, we require h, t and S to meet the following conditions:

- For CS and VP, the weights inequality $w_h \geq \sum_{s \in S} t_s w_s$ must hold.
- For BP, we use the criterion $w_h \geq \sum_{s \in S} w_s$.
- For VC, h and S need to fulfill $N(h) \cup \{h\} \supseteq \bigcup_{s \in S} N(s)$, and S must constitute an independent set.
- For BPC, we require both of the above conditions of BP and VC.

4.1. Dynamic Separation of Violated Dual Inequalities

The careful dynamic generation and addition of violated WSIs prevents the RMP from being stuffed with useless DOIs as it may happen with an a-priori addition of many DIs that one suspects to help stabilizing the CG process. Clearly, mixing both strategies, the a-priori integration of expectedly helpful DIs and the dynamic generation of violated DIs, is a viable strategy that we analyze later in the computational experiments.

For the identification of violated WSIs, a separation procedure is needed. Let $\bar{\pi}_i, i \in I$ be the dual values in a CG iteration after the reoptimization of the RMP. (We need to distinguish between a specific dual solution $\bar{\pi}$ and the dual variables π.) The task of the separation procedure is to identify one or several violated WSIs $(h \leftarrow t, S)$ if any, i.e., with $\bar{\pi}_h < \sum_{s \in S} t_s \bar{\pi}_s$. In the following, we describe the individual separation algorithms that we later use for CS, BP, VC, and BPC in our computational study.

Cutting Stock. For CS, we consider a given item $h \in I$ for which a violated WSI of the form $(h \leftarrow t, S)$ requires the identification of the subset S and the quantities $t_s \in \mathbb{Z}^S_{>0}$. Such a violated WSI exists if and only if $\bar{\pi}_h < z_h$, where $z_h = \max_{i \in I} \bar{\pi}_i t_i$ such that $\sum_{i \in I} w_i t_i \leq w_h$. This is, for each item $h \in I$, an UKP with identical coefficients as in the pricing problem, except for a smaller capacity of w_h instead of L. Note that the WSI $(h \leftarrow t, S)$ with t and $S = \{i \in I : t_i > 0\}$ for the optimal solution to the UKP above is the most violated WSI for this specific item h.

From a worst-case point of view, the best known algorithms for solving the UKP pricing problem are based on dynamic programming (DP), and they require $\mathcal{O}(mL)$ time (see Kellerer *et al.*, 2004). Figure 1 shows the state graph, in which states correspond with residual capacities $0, 1, 2, \ldots, L$. For each item $i \in I$, arcs $(p, p + w_i)$ are present for $0 \leq p$ and $p + w_i \leq L$. They all have an associated profit $\bar{\pi}_i$. Additional arcs $(p, p + 1)$ with profit 0 model possible slack in a solution. UKP can now be interpreted as a longest 0-L-path problem in this directed acyclic graph. When solved with DP, the value z_h for $h \in I$ can be found at state w_h. If $\bar{\pi}_h < z_h$, then the associated longest 0-w_i-path defines the subset of selected items $S \subseteq I$ and their quantities $t \in \mathbb{Z}^S_+$. Thus, the exact dynamic separation of violated WSIs is a by-product of solving the pricing problem for CS. The additional effort for identifying a most violated WSI is $\mathcal{O}(m)$ and, therefore, the separation comes almost for free. Note that the approach provides not only a most violated WSI, but several other violated WSIs can also be separated.

For VP, the multi-dimensional extension of CS, the pricing problem is an unbounded DKP (Kellerer *et al.*, 2004, Chapter 9). Finding a most violated WSI requires the solution of m smaller DKPs, which are identical to the pricing problem, but the multi-dimensional capacity $(L_1, \ldots, L_d)$ is set to $(w_{h1}, \ldots, w_{hd})$ for each $h \in I$. The literature reports different exact solution algorithms for DKP (see Ozden, 1988; Fréville, 2004). It seems that none of these approaches allows the direct retrieval of the solution to the WSI separation problem in a straightforward way.

Bin Packing. The pricing problem and the SI separation problem for BP are KPs. Again, they again can be solved efficiently in $\mathcal{O}(mL)$ pseudo-polynomial time using dynamic programming (Kellerer *et al.*, 2004, Chapter 5). The state graph however differs from the one of UKP. There is one stage for each item $i \in I = \{1, \ldots, m\}$ plus one start stage with the single state 0. At stage $i \in I$, the states $0, 1, 2, \ldots, L$ are the possible residual capacities. Stage $i-1$ and stage i for $i \in I$ are connected in the following way: The arcs $(p, p + w_i)$ with profit π_i for $0 \leq p \leq p + w_i \leq L$ ($p = 0$ for $i = 1$) model the inclusion of item i in the respective solution, while the arcs (p, p) with profit 0 for $0 \leq p \leq L$ ($p = 0$ for $i = 1$) model the exclusion of item i. Using forward DP, the values at the states w_h at stage m are $z_h = \max_{i \in I} \bar{\pi}_i t_i$ such that $\sum_{i \in I} w_i t_i \leq w_h$ and $t \in \{0, 1\}^I$, i.e., the values of the same KP instance as the pricing problem, but for smaller residual capacities. For each $h \in I$, the corresponding path from 0 to w_h (connecting stages 0 and m) defines a set S of items to include. The respective SI $(h \leftarrow S)$ is violated if and only if $z_h > \bar{\pi}_h$ holds. Summing up, as for CS, if the BP pricing problem is already solved with DP, the additional effort needed to identify violated SIs is $\mathcal{O}(m)$. Thus, separation is a by-product of pricing.

Vertex Coloring. For VC, we consider again a fixed vertex $h \in I$. For some $S \subseteq I \setminus \{h\}$, $(h \leftarrow S)$ is a possible SI if S is an independent set and $N(h) \cup \{h\} \supseteq N(s)$ holds for all $s \in S$. Denote by $S_h = \{s \in I : N(h) \cup \{h\} \supseteq N(s)\}$ the set of all vertices that have a subset of the conflicts of vertex h. Then, the SI $(h \leftarrow S)$ is violated if and only if $\bar{\pi}_h < z_h$, where $z_h = \max_{S \subseteq S_h} \sum_{s \in S} \bar{\pi}_s$ such that S is an independent set. This is a MWIS problem (Östergård, 2002). Again, the separation problem is identical to the pricing problem, but defined on the vertex-induced subgraph $G[S_h]$. As such, it is strongly $\mathcal{NP}$-hard and no by-product of solving the pricing problem.

For the separation of violated SIs $(h \leftarrow S)$, we solve a MWIS for each vertex $h \in I$ using the same exact algorithm as for the pricing problem, see Section 5. The respective sets S_h for each vertex $h \in I$ are computed once and prior to the CG process. Typically, the sets S_h are very small compared to I in the VC benchmark instances. As a result, separation times are negligible. Analog to CS and BP, the separation procedure provides not only the most violated SI, but also several others.

Bin Packing with Conflicts. Since the BPC can be seen as the synthesis of BP and VC, it can be expected that ideas for the separation of DIs can be adapted from these problems. First note that the CG subproblem is a binary *knapsack problem with conflicts* (KPC). For general conflicts, i.e., defined by arbitrary conflict graphs $(I, \mathcal{E})$, the KPC is strongly $\mathcal{NP}$-hard and also practically very hard to solve (Hifi and Michrafy, 2007; Bettinelli *et al.*, 2014). If the conflict graph is of bounded treewidth or is a chordal graph, an efficient DP algorithm with complexity $\mathcal{O}(m^2 L)$ has been proposed by Pferschy and Schauer (2009). The CG literature on BPC focuses on an even more restricted class of conflict graphs, namely interval graphs. Herein, each item $i \in I$ has an associated interval $\mathcal{I}_i := (a_i, b_i)$, and two items $i, j \in I$ are in conflict if and only if $\mathcal{I}_i \cap \mathcal{I}_j \neq \emptyset$. For interval graphs, the KPC subproblem can be solved by dynamic programming in $\mathcal{O}(mL)$ time and space with the algorithm of Sadykov and Vanderbeck (2013). The following computational analysis is likewise restricted to the case of interval conflict graphs.

We briefly sketch the DP algorithm for the KPC proposed by Sadykov and Vanderbeck (2013): The state graph consists of the same $\mathcal{O}(mL)$ states as the one for KP, one for each residual capacity $0, 1, 2, \ldots, L$ and item $i \in I$ plus one start state at stage 0. The crucial difference, however, is that items $i \in I$ need to be sorted according to the rhs b_i of their conflict intervals. Let $pred_i$ denote the a last vertex in $\{0, 1, 2, \ldots, i-1\}$ that is not in conflict with a vertex i (items are sorted in the above order and 0 is an artificial start item not in conflict to any other). Then, the arcs $(p, p + w_i)$ connect stage $pred_i$ to stage i (instead of stages $i-1$ and i as for KP). They represent the selection of the item i and result in the profit $\bar{\pi}_i$. Similar to KP,

the arcs (p, p) connect stages $i-1$ and i. They model that item i is not part of the KPC solution with resulting profit of 0. It is easy to see that by construction paths in the state graph correspond to conflict free packings.

As before, for the separation of SIs $(h \leftarrow S)$, we consider the possible items $h \in I$ one by one. The most violated SI $(h \leftarrow S)$ results from the solution of a smaller KPC subproblem, which is defined by the smaller capacity w_h $(\leq L)$ and on the item subset $S_h = \{s \in I : w_h \geq w_s, N(h) \cup \{h\} \supseteq N(s)\}$. This separation problem is *no* by-product of the BPC subproblem in this case. In fact, the DP value for the last item m and the residual capacity w_h results in a solution, which is an independent set S with weight not exceeding w_h. However, the necessary condition $S \subseteq S_h$ does generally not hold. Thus, the DI $(h \leftarrow S)$ does typically not qualify as a possible SI.

Since separation is non-trivial, the following three procedures for identifying violated SIs will be considered:

pairs Pair inequalities can be separated by explicit inspection, which takes $\mathcal{O}(m^2)$ time.

DP Based on the solution of the KPC pricing problem, the following DP-based heuristic can be applied: One inspects all states $w = 1, 2, \ldots, L$ at the final stage m. Each state reached represents a solution, which is an independent set S_w with weight w. For this set S_w, one can find a best item $h \in I$ (if any) with $w_h \geq w$ and $S_h \supseteq S_w$ so that h and S_w induce a SI. Section B in the Appendix explains that the entire procedure can be implemented to run in $\mathcal{O}(mL)$ time.

exact The exact separation of SIs $(h \leftarrow S)$ must solve a smaller KPC for each vertex $h \in I$ with capacity w_h and vertices $s \in S_h$. This requires $\mathcal{O}(m^2 L)$ time, which will turn out to be rather time consuming. Thus, we only solve the DP for an item $h \in I$, for which the LP-bound of the KP with vertices $s \in S_h$ results in a sufficiently high violation compared to the currently most violated SI found. The procedure is still exact, but potentially misses some SIs when separating multiple SIs within certain quality compared to the most violated one.

4.2. Over-Stabilization and Recovery Feasible Primal Solutions

For further stabilizing the CG process, we propose to not only use DIs that are proven to be DOIs or DDOIs. It may then happen that all dual-optimal solutions π^* to D are cut off. From a primal perspective, this means that we have over-stabilized the CG process and that we terminate with a primal solution to $\tilde{P}$ that cannot be transformed into a feasible solution of P with the same cost, see Proposition 1. The following example illustrates this behavior.

Example 1. Consider an instance of BP with bin capacity $L = 10$ and four items with $w_1 = 5$, $w_2 = 2$, $w_3 = 2$, and $w_4 = 2$. The linear relaxation uses the following four feasible packings $(1,1,1,0)^\top$, $(1,1,0,1)^\top$, $(1,0,1,1)^\top$, and $(0,1,1,1)^\top$ with corresponding dual constraints $\pi_1 + \pi_2 + \pi_3 \leq 1$, $\pi_1 + \pi_2 + \pi_4 \leq 1$, $\pi_1 + \pi_3 + \pi_4 \leq 1$, and $\pi_2 + \pi_3 + \pi_4 \leq 1$, respectively. The unique primal and dual optimal solutions are $\lambda^* = \pi^* = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3}, \frac{1}{3})^\top$ with $z(P) = z(D) = \frac{4}{3}$.

After adding the SI $\pi_1 \geq \pi_3 + \pi_4$, the optimal solution π^* is cut off, and a new dual-optimal solution is given by $(\frac{1}{2}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4})^\top$ with $z(\tilde{D}) = 1.25$. Thus, the SI $\pi_1 \geq \pi_3 + \pi_4$ is no DDOI. The corresponding column in $\tilde{P}$ is $(-1, 0, 1, 1)^\top$, and an optimal solution for $\tilde{P}$ is given by $(\tilde{\lambda}^*, y^*) = ((\frac{1}{2}, \frac{1}{2}, \frac{1}{4}, 0)^\top, \frac{1}{4})$.

Transforming $(\tilde{\lambda}^*, y^*)$ into a feasible solution to P means exchanging item 1 by items 3 and 4 in one of the chosen bin columns that include item 1. Doing so, however, results in a new bin column, in which either item 3 or 4 are packed twice, which is infeasible for BP.

Note that for CS, $(\tilde{\lambda}^*, y^*)$ can be transformed into a feasible solution for P because columns are allowed to have non-negative integer coefficients. This also reflects the intuition that WSIs are DOIs for CS and VP, while they are not for BP.

Before we formally state the recovery algorithm, we introduce some additional notation. Recall that any WSI is of the form $(h \leftarrow t, S)$, where $h \in I$ is the row with the -1 entry in the coefficient matrix E and (t, S) are the non-zero entries $t_s > 0$ for $s \in S$. Let K denote the row indices of E. Thus, in the primal model $\tilde{P}$, these indices refer to the variables $y_k, k \in K$ and columns of $E^\top$ with associated WSI $(h \leftarrow t, S)$.

For abbreviation, we write $k = k(h \leftarrow t, S)$ in this case. Two indices $j \in J$ and $k \in K$ are said to be *WSI compatible* if $a_j - u_h + t \in A$ for $h \in I$ defined by $k = k(h \leftarrow t, S)$.

Algorithm 1: Column generation recovery algorithm

1 **repeat**
2 Compute solution (λ^*, y^*) to $\tilde{P}$ using CG
3 **while** *WSI compatible pairs* $(j,k) \in J \times K$ *with* $\lambda^*_j > 0$ *and* $y^*_k > 0$ *exist* **do**
4 Select a WSI compatible pair $(j,k) \in J \times K$ with $\lambda^*_j > 0$ and $y^*_k > 0$
5 Let $\delta := \min\{\lambda^*_j, y^*_k\}$
6 Let $j' \in J$ such that $a_{j'} = a_j - u_h + t$
7 Let $\lambda^*_j := \lambda^*_j - \delta$, $y^*_k := y^*_k - \delta$, $\lambda^*_{j'} := \lambda^*_{j'} + \delta$
8 **if** $y^* > 0$ **then**
9 Select one (or several) $h \in I$ such that $y^*_k > 0$ for h defined by $k = k(h \leftarrow t, S)$
10 Eliminate all WSIs $(h \leftarrow t', S')$ for $S' \subseteq I, t' \in \mathbb{Z}^I_{>0}$ from the RMP and prohibit their re-generation
11 **until** $y^* = 0$
Result: Solution λ^* to P

The recovery algorithm is formally stated by Algorithm 1. First note that by definition of compatible pairs $(j,k) \in J \times K$ the value of δ is strictly positive in Step 5. Moreover, the assignments in the Step 7 do neither invalidate the primal feasibility $A\lambda^* + E^\top y = b$ nor alter the cost of the solution. This results from the equalities $a_{j'} = a_j - u_h + t$ and $c_j = c_{j'} = 1$. Therefore, the inner loop (Steps 4–10) is successful if an equivalent primal solution with $y^* = 0$ is constructed. Otherwise (Step 8), there must exist an active WSI with row index h for which no compatible $j \in J$ exists. In this case, replacements to row h cannot be recovered. The recovery algorithm has failed in this iteration, and therefore some active WSIs are eliminated from the RMP and their re-introduction is made impossible. As a result, the next RMP solution (Step 2) does not contain any WSIs that refer to the row index $h \in I$ selected in Step 9. Thus, after a maximum of m outer loops, the Algorithm 1 must terminate with a solution $(\lambda^*, 0)$ to $\tilde{P}$. Since the recovery procedure is cost preserving, Proposition 1(v) is fulfilled so that the modified λ^* is an optimal primal solution to P.

The Steps 4 and 9 leave different strategies for choices of specific pairs (j,k) and h open. For Step 4, the selection of a pair (j,k) leading to a maximal δ may help to minimize the number of necessary iterations of Algorithm 1. For the selection of a row h in Step 9, we observed in our computational test that the number of possible choices is always small so that we always choose all $h \in I$ with $y^*_k > 0$ and $k(h \leftarrow t, S)$.

5. Computational Results

In the following, we present computational results for CS, BP, VC, and BPC showing how the stabilization with DIs affects the CG process. Furthermore, we analyze the effects of using different classes of DIs and show that the dynamic generation of DIs is often beneficial. All algorithms described in this paper were implemented in C++ using CPLEX 12.5 as LP solver. The experiments were conducted on two standard PCs, one with an Intel(R) Core(TM) i7-3770 (for BP) and one with an Intel(R) Core(TM) i7-2600 (for CS, VC, and BPC), each running at 3.4 GHz and with 16.0 GB main memory using a single thread only.

5.1. Results for Bin Packing

We use the same basic CG algorithm for each of the different CG strategies that we compare. The main features of the CG algorithm are as follows: We use a *best fit decreasing* heuristic (Martello and Toth, 1990) to generate an initial set of columns for warm starting the CG process. The subproblems are solved by the DP algorithm described in Section 4.1 and a single best reduced-cost bin column is generated in each iteration. Pre-tests indicated that adding a single bin column is generally the best performing strategy. If DIs are generated dynamically, multiple SI columns with a violation of at least 25% of the most violated SI are added. In preliminary computational tests, this appeared to be marginally superior compared to values of 50% and 75% and adding a single DI column only.

The first strategy we consider is the standard CG approach without using any DIs referred to by *standard* in the following. In contrast, the strategy denoted by *aggregation* uses constraint aggregation on items with identical weight. This is the approach followed by Ben Amor *et al.* (2006) and can be seen as the state of the art for BP. It is, therefore, used as the baseline strategy for comparisons with all other strategies. Moreover, all following strategies also make use of this aggregation. We also consider using DIs in a static fashion only (*static*), i.e., the addition of a set of expectedly helpful DIs to the RMP as static columns prior to the CG process. Motivated by preliminary computational tests, we use DIs that are similar to what Ben Amor *et al.* (2006) proposed for CS. More precisely, we use the $m-1$ ranking inequalities, which are DDOIs for BP, see Theorem 2. Additionally, $\mathcal{O}(m)$ SIs $(h \leftarrow S)$ with $|S| = 2$ are added, namely one for each item $h \in I$. Herein, the set $S = \{i, j\}$ is chosen (if existent) such that the slack in $w_h \geq w_i + w_j$ is minimal (and non-negative). The pure dynamic generation of violated SIs using the separation procedure of Section 4.1 is denoted by *dynamic*. Finally, we consider the combination of the latter two, i.e., the use of static DIs and the dynamic generation of additional violated SIs. This last strategy is referred to as *stat+dyn*.

As test instances we used the benchmark sets by Scholl *et al.* (1997) and Sim and Hart (2013) comprising a total of 1,210 and 15,830 instances, respectively. Both benchmark sets are subdivided into several subclasses differing with respect to the number of items, the capacity, and the variance of the item weights resulting in instances with very different characteristics and degrees of complexity. In the following, we present results averaged over all subclasses. However, we include only those instances, for which the computation time of *aggregation* is at least one second. This reduces the number of benchmark instances to 207 and 4,032, respectively. The other instances are solved in too little time with the stronger algorithmic strategies. Hence, the consideration of these instances would, if they were taken into account, produce unreliable values for the computation times.

We further differentiate two subgroups for each of the benchmark sets: Groups 1 comprise the instances with computation times between one and ten seconds for *aggregation*, and it consist of 157 and 3,352 instances for the sets of Scholl *et al.* (1997) and Sim and Hart (2013), respectively. The hardest instances in terms of solution time (> 10 seconds for *aggregation*) are in groups 2 consisting of the remaining 50 and 680 instances.

Table 2 summarizes the results for BP, where the table columns have the following meaning:

m	The number of rows in the RMP relative to *aggregation*
Solution time	The time for solving the linear relaxation of the RMP relative to *aggregation*; we present average, maximum, and minimum values over the instances
# Iterations	The number of iterations relative to *aggregation*; we present average, maximum, and minimum values over the instances
SP/RMP	The ratio of the times spent for solving the subproblem and re-optimizing the RMP, averaged over the instances.
# Dynamic DIs	The number of DIs that are generated dynamically during the CG process; we present average, maximum, and minimum values over the instances
Recovery	The total number of instances for which a recovery is needed (# inst.) and the maximum number of iterations of the recovering algorithm for one of the instances (max it.).

Table 2 shows that the dynamic generation of DIs results in an average speedup of approximately factor two compared to *aggregation*. For the hard group 2 instances, the average speedup even reaches factor three to four. Thereby, *stat+dyn* is slightly faster than the pure use of dynamic DIs. There are also instances for which *stat+dyn* and *dynamic* perform worse than *aggregation* in terms of computations times. However, computation times never go beyond 139% and 130% of *aggregation* for *dynamic* and *stat+dyn*, respectively. The biggest speedups on the contrary exceed factor 15, while *standard* CG is at least one order on magnitude slower than all strategies applying *aggregation*.

The static use of DIs results only in a small speedup of approximately $2-3\%$ on average. Recall that ranking inequalities were not yet known as DDOIs in the work of Ben Amor *et al.* (2006). Therefore, a static DI-based CG algorithm was not considered for BP at that time.

Regarding the number of iterations, the differences between the strategies using dynamic DIs and *aggregation* are even more pronounced than for the computations times. Also *static* performs considerably better

as it requires only about two thirds of the iterations of *aggregation*.

Another interesting result is that a recovery is necessary only for a very small number of instances. Even more, the maximum number of iterations of the recovery algorithm is two (cf. Algorithm 1). This demonstrates that the SIs we add to the RMP (static or dynamic) are indeed DDOIs for most of the instances in the benchmark.

Instances	Algorithm	m	Solution time (avg./max/min)	# Iterations (avg./max/min)	SP/ RMP	# Dynamic DIs (avg./max/min)	Recovery (# inst./max it.)
Sim and Hart All instances ($n = 4{,}032$)	*standard*	3.1	5.00/31.46/0.84	3.45/20.47/0.96	5.8	–	–/–
	aggregation	1	1.00/1.00/1.00	1.00/1.00/1.00	31.7	–	–/–
	static	1	0.97/1.39/0.67	0.66/1.02/0.47	44.3	–	7/1
	dynamic	1	0.54/1.39/0.07	0.43/1.11/0.08	21.3	1462/9852/26	54/2
	stat+dyn	1	0.52/1.30/0.06	0.36/1.02/0.03	26.9	864/7351/0	54/2
Sim and Hart Group 1 ($n = 3{,}352$)	*standard*	3.3	5.35/31.45/0.87	3.73/20.47/0.96	2.9	–	–/–
	aggregation	1	1.00/1.00/1.00	1.00/1.00/1.00	30.1	–	–/–
	static	1	0.96/1.35/0.67	0.65/1.02/0.47	41.7	–	7/1
	dynamic	1	0.57/1.39/0.09	0.45/1.11/0.10	20.1	1007/7029/26	37/2
	stat+dyn	1	0.55/1.30/0.08	0.39/1.02/0.04	23.2	529/5531/0	39/2
Sim and Hart Group 2 ($n = 680$)	*standard*	2.0	3.30/8.55/0.84	2.06/3.70/1.05	20.5	–	–/–
	aggregation	1	1.00/1.00/1.00	1.00/1.00/1.00	38.9	–	–/–
	static	1	1.03/1.39/0.79	0.72/0.87/0.60	56.0	–	0/0
	dynamic	1	0.36/1.21/0.07	0.30/0.81/0.08	26.7	3074/9852/669	17/2
	stat+dyn	1	0.35/1.20/0.06	0.24/0.81/0.03	43.5	2520/7351/0	15/2
Scholl *et al.* All instances ($n = 207$)	*standard*	3.0	5.22/28.78/0.91	3.48/20.26/0.96	4.9	–	–/–
	aggregation	1	1.00/1.00/1.00	1.00/1.00/1.00	29.6	–	–/–
	static	1	0.98/1.22/0.76	0.66/1.00/0.47	41.3	–	1/1
	dynamic	1	0.41/1.07/0.08	0.35/1.03/0.10	15.9	2252/11567/35	6/2
	stat+dyn	1	0.39/1.12/0.07	0.26/1.00/0.04	21.4	1465/7270/0	2/1
Scholl *et al.* Group 1 ($n = 157$)	*standard*	3.3	5.83/28.77/1.08	3.96/20.26/1.06	2.7	–	–/–
	aggregation	1	1.00/1.00/1.00	1.00/1.00/1.00	32.1	–	–/–
	static	1	0.97/1.19/0.76	0.65/1.00/0.47	42.8	–	1/1
	dynamic	1	0.45/1.07/0.13	0.38/1.03/0.14	17.1	1372/4472/35	4/2
	stat+dyn	1	0.43/1.12/0.09	0.29/1.00/0.07	20.9	871/3443/0	2/1
Scholl *et al.* Group 2 ($n = 50$)	*standard*	1.9	3.29/8.11/0.91	1.99/3.57/0.96	11.8	–	–/–
	aggregation	1	1.00/1.00/1.00	1.00/1.00/1.00	22.4	–	–/–
	static	1	0.99/1.22/0.86	0.68/0.79/0.55	37.1	–	0/0
	dynamic	1	0.29/1.07/0.08	0.24/0.70/0.10	12.4	5015/11567/936	2/1
	stat+dyn	1	0.27/1.08/0.07	0.17/0.70/0.41	22.8	3329/7270/0	0/0

Table 2: Computational results for the bin packing problem (BP)

Table 2 also indicates that there are instances, for which no dynamic DIs are generated when static DIs are added a priori. These are instances in which the item sizes are very similar so that no three items $h, i, j \in I$ with $w_h \geq w_i + w_j$ exist. It is therefore clear that no SIs ($h \leftarrow S$) with $|S| \geq 2$ exist in these cases so that no additional speedup can be achieved. Our results include a total of 1,427 and 39 such instances for the sets of Sim and Hart and Scholl *et al.*, respectively. In summary, the overall average speedup for the instances, in which SIs with $|S| \geq 2$ exist, actually exceeds factor two significantly.

In Figure 2, we display more generally the influence of the number of possible SIs on the performance of the different CG strategies. The ratio $R_{BP} = \max_{i \in I} w_i / \min_{i \in I} w_i$ serves as an approximate measure of how many SIs potentially exist. Clearly, this ratio reveals no information on the distribution of the items weights. In fact, the number of SIs is generally not computable, but R_{BP} can be determined easily for each BP instance. Note that a ratio $R_{BP} < 2$ reflects the case described above, in which no SI with $|S| \geq 2$ exist. The relationship between the ratio R_{BP} and the speedups using DIs is depicted in Figure 2: It shows the average computation times of *static*, *dynamic*, and *stat+dyn* relative to *aggregation* for all 4,032 considered Sim and Hart instances depending on R_{BP}. Note that the abscissae groups instances according to R_{BP} (rounded to one decimal) and that the ordinate shows the relative computation times in logarithmic scale.

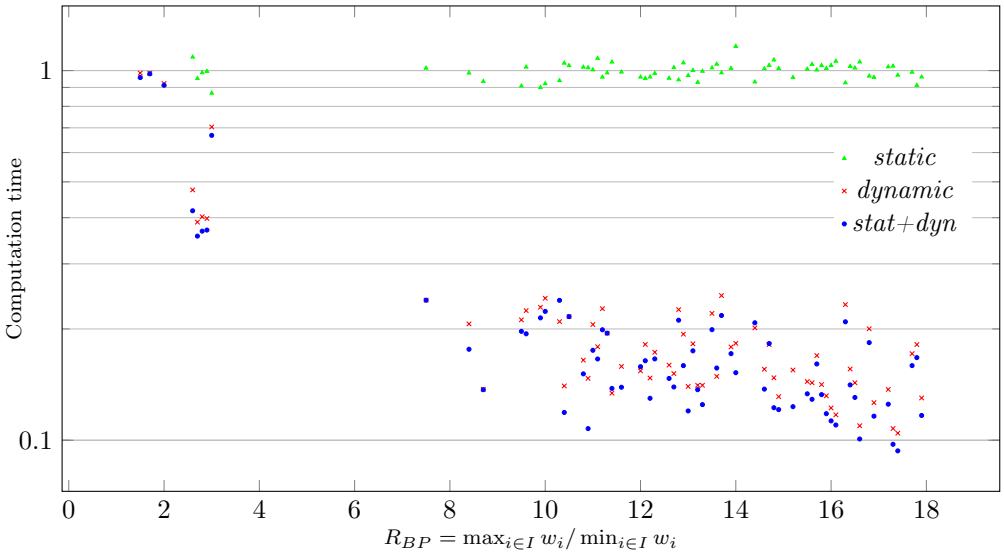

Figure 2: Average computations times relative to *aggregation* (ordinate) depending on the ratio $\max_{i \in I} w_i / \min_{i \in I} w_i$ (abscissae) for all considered Sim and Hart instances for the bin packing problem (BP)

Figure 2 clearly demonstrates that the dynamic generation of SIs, i.e., implicitly considering the set of all SIs, works better the bigger the ratio R_{BP}. One can therefore suspect that there is a highly significant positive correlation between the number of SIs and the resulting acceleration of the CG process. For larger values of R_{BP}, average speedups reach factors between five and ten.

5.2. Results for Cutting Stock

For CS, the basic CG algorithm and the different stabilization strategies are similar to BP. We only describe the differences. The UKP subproblems are solved with the DP approach of Section 4.1. For CS, the basic CG algorithms already makes use of constraint aggregation. Thus, the strategies *standard* and *aggregation* coincide, and results are solely reported as *standard*. Strategy *static* as proposed by Ben Amor *et al.* (2006) is the state of the art for CS. Therefore, *static* is the baseline strategy meaning that results for all other strategies are presented relative to it. Recall that the WSIs are DOIs for CS and no recovery is needed, see Theorem 1(i).

Generally, the BP benchmark instances of the previous section can also be interpreted as CS instances, in which items of identical length need to be aggregated. In preliminary experiments, we found that the UKP subproblems of CS are solved significantly faster than the (binary) KP subproblems of BP for all the benchmark instances. We suspect that this is due to the smaller state space of the DP of the UKP compared to the one of KP. Recall that the state graph of the UKP (depicted in Figure 1) has only $L+1$ states, while the state graph of KP has $1+(L+1)m = \mathcal{O}(mL)$ states. We would like to stress that for both UKP and KP we implemented state-of-the-art DP algorithms: In particular, for KP we do not explicitly build the $\mathcal{O}(mL)$ states, but use a list-based implementation as discussed in (Kellerer *et al.*, 2004, Sect. 3.4). In contrast, for UKP we found that a straightforward array-based implementation of the DP approach is faster than the list-based approach. We suspect that on a modern CPU, the smaller state graph of UKP can be accessed much faster (due to caching techniques) so that the solution of the UKP subproblems as they occur in the BP benchmark instances is possible in almost no (measurable) time.

As a result, of the 17,040 benchmark instances from the sets of Scholl *et al.* and Sim and Hart over 99% are solved by *standard* CG in less than 1 second of computation time, most of them in only a few milliseconds, meaning that a large number of UKP can be solved in this short time. Therefore, we considered these CS instances of limited interest and generated new and harder CS instances. These are characterized by huge values for the capacity (in order to complicate the subproblems) and larger numbers of items with distinct

lengths. The instances we generated can be grouped into different subclasses of problems. Capacities take values $L \in \{500{,}000, 1{,}500{,}000\}$ and integer item lengths are uniformly drawn from $[2/15\,L, 2/3\,L]$ and $[1/150\,L, 2/3\,L]$ resulting in items for which the ratios w_i/L are analog to those in the benchmark instances used by Ben Amor *et al.* (2006). The number of items with distinct item length takes values $m = \{125, 250, 500\}$. Demands are uniformly distributed in the range $[1, 20]$. For each subclass, 20 instances were generated. The entire benchmark set comprises 240 instances and is available at `http://logistik.bwl.uni-mainz.de/Dateien/CS.zip`.

Table 3 presents our results for CS. First and foremost, the overall speedups of using stabilized CG are significantly smaller for CS compared to BP. The *static* use of DIs results in an average speedup of about one third. The additional use of dynamically generated DI saves an extra 17% of the computation time. The pure *dynamic* generation even results in a slightly slower overall algorithm than the *static* version.

The results regarding the number of iterations are more in favor of using stabilized CG. Iterations are on average reduced by a larger factor than the computation times. The results by subclasses are similar to the overall results as can be seen in Section C of the Appendix.

Instances	Algorithm	Solution time (avg./max/min)	# Iterations (avg./max/min)	SP/ RMP	# Dynamic DIs (avg./max/min)
All instances ($n = 240$)	*standard*	1.50/8.73/0.93	1.77/29.00/1.31	17.6	–
	static	1.00/1.00/1.00	1.00/1.00/1.00	13.7	–
	dynamic	1.03/4.80/0.44	0.86/9.00/0.41	6.5	5590/18100/247
	stat+dyn	0.83/1.53/0.29	0.75/1.00/0.31	9.8	1120/3742/0

Table 3: Computational results for the cutting stock problem (CS)

5.3. Results for Vertex Coloring

Analog to BP and CS, we use one basic CG algorithm for our experiments and run it with different strategies regarding the stabilization with DIs. To warm start the CG process, a simple greedy coloring is performed. The MWIS pricing problem is solved using the algorithm and its implementation by Held *et al.* (2012). In each iteration, a single best reduced-cost column is added to the RMP. If dynamic generation of DIs is used, violated SIs are separated using the procedure described in Section 4.1 and multiple SI columns with a violation of at least 25% of the most violated SI are generated.

Standard denotes the basic CG algorithm without stabilization. It serves as the baseline algorithm for the other strategies. The algorithm using constraint elimination is denoted by *elimination*, and all following strategies also make use of constraint elimination. For an approach that uses DIs in a static fashion, recall that no linear system of DIs can impose all pair inequalities, see Section 3.3. Still, we suggest a strategy that uses a linear-sized subset of the pair inequalities similar to the ranking inequalities in CS and BP. For each vertex $h \in I$, we determine a vertex $i = \operatorname{argmax}_{s \in S_h} |N(s)|$. This is the vertex in the candidate set S_h that has the most conflicts, and we add the corresponding SI column to initialize the RMP. The intuition is that this vertex i is the hardest one to cover of all the vertices that are easier to cover than h. Therefore, the repetition of this SI selection for each vertex $h \in I$ should result in a near ranking of all the duals. We refer to this approach as *static*, and no SIs with $|S| > 1$ are used in this case. The strategies *dynamic* and *stat+dyn* are analog to CS and BP.

Note that because constraint elimination is performed in each strategy that uses DIs, the candidate sets S_h for all vertices $h \in I$ comprise only vertices $s \in S_h$ for which $h \in N(s)$ holds. Consequently, by Theorem 2(b) all SIs generated by our separation procedure are in fact DOIs and no recovery is needed.

As benchmark problems, we used the instances from the graph coloring benchmark web page `https://sites.google.com/site/graphcoloring/home` of Gualandi and Chiarandini (2014). The entire benchmark comprises 136 instances. We restrict our analysis to those instances in which at least one possible SI exists. This eliminates approximately half of the instances so that 74 VC instances remain. Furthermore, we exclude from our computational analysis those instances, for which the linear relaxation of the RMP cannot be solved within a time limit of one hour using all strategies (15 instances), and those instances, for which

Instances	Algorithm	m	Poss. repl. %	Solution time (avg./max/min)	# Iterations (avg./max/min)	SP/ RMP	# Dynamic DIs (avg./max/min)
All instances ($n = 25$)	*standard*	1	0	1.00/1.00/1.00	1.00/1.00/1.00	29.8	–
	elimination	0.80	0	0.88/1.16/0.35	0.93/1.12/0.52	33.0	–
	static	0.80	0.64	0.57/1.17/0.06	0.71/1.01/0.15	13.8	–
	dynamic	0.80	0.64	0.54/0.96/0.03	0.72/1.04/0.08	11.5	349/5314/0
	stat+dyn	0.80	0.64	0.53/0.91/0.06	0.70/0.99/0.08	12.0	121/2015/0
Group 1 ($n = 11$)	*standard*	1	0	1.00/1.00/1.00	1.00/1.00/1.00	59.7	–
	elimination	0.96	0	1.01/1.16/0.90	1.00/1.12/0.88	62.0	–
	static	0.96	1.45	0.35/0.78/0.06	0.60/0.90/0.15	17.5	–
	dynamic	0.96	1.45	0.33/0.71/0.04	0.57/0.92/0.08	13.9	765/5314/26
	stat+dyn	0.96	1.45	0.31/0.70/0.06	0.55/0.90/0.08	15.1	270/2015/0
Group 2 ($n = 14$)	*standard*	1	0	1.00/1.00/1.00	1.00/1.00/1.00	6.3	–
	elimination	0.67	0	0.78/1.04/0.35	0.87/1.03/0.52	10.3	–
	static	0.67	0.01	0.74/1.17/0.27	0.81/1.01/0.43	10.8	–
	dynamic	0.67	0.01	0.71/0.96/0.40	0.81/1.04/0.45	9.6	23/288/0
	stat+dyn	0.67	0.01	0.70/0.91/0.42	0.81/0.99/0.43	9.6	4/52/0

Table 4: Computational results for the vertex coloring problem (VC)

the solution time is less than 0.1 seconds for the *standard* strategy (34 instances). This leaves 25 instances for the test set which we further subdivided into two groups: Group 1 comprises all instances for which the ratio $R_{VC} = (\sum_{h \in I} |S_h|)/m^2 > 0.001$, while the remaining instances form group 2. R_{VC} measures the number of possible replacements of items relative to the instance size, thus, giving an approximation on the number of possible SIs relative to the instances size. One can expect that (dynamic) stabilization generally performs better for group 1 than for group 2.

The results of the computational tests are summarized in Table 4. The additional column *Poss. repl. %* shows the ratio R_{VC} in percent. A general observation is that with stabilized CG an average speedup of approximately factor two can be achieved compared to *standard*. Thereby, all three strategies using DIs perform comparably well, although the biggest gains can be achieved with *stat+dyn*. For larger values of R_{VC}, stabilization with SIs is even more attractive and the average speedup reaches factor three for group 2. Table 4 also reveals that stabilization comes almost for free, since the maximum relative computation time for *static* is only 17% longer than *standard* and the strategies with dynamic separation of SIs never take more computation time than *standard*. On the other hand, the biggest speedups exceed factors 15, 30, and 15 for *static*, *dynamic*, and *stat+dyn*, respectively. Expectedly, *elimination* works better the more vertices can be eliminated. The insights regarding the number of iterations are evident: stabilization by DIs can help to reduce the number of CG iterations and is rarely detrimental.

5.4. Results for Bin Packing with Conflicts

Analog to Sections 5.1–5.3, we run one basic CG algorithm for BPC with different stabilization strategies to analyze their impact on the CG approach. An initial set of columns to warm start the process is obtained by performing a *first fit decreasing* heuristic several times with different orderings of the items. The KPC subproblems, for which the conflicts are defined on interval graphs in all benchmark instances, are solved with the DP algorithm of Sadykov and Vanderbeck (2013) described in Section 4.1. Again, a single best reduced-cost bin and multiple SI columns with a minimum violation of 25% of the most violated SI are generated in each iteration.

Standard CG without stabilization serves as the baseline strategy for comparisons with the other strategies. Similar to VC, no linear system of DIs can impose all pair inequalities, see Section 3.3. For an approach that uses DIs in a *static* fashion, we therefore follow a strategy using the same basic idea as for VC: For each item $h \in I$ we identify the item $i \in S_h$ (if any) that is the hardest one to cover of all the vertices that are easier to cover than h and add the respective pair inequality to the RMP. For BPC, however, the hardness to cover an item i depends on both its weight w_i and its conflicts $N(i)$. Therefore, as item i we choose one that maximizes the sum of w_i/w_h and $|N(i)|/|N(h)|$. SIs with $|S| > 1$ are not used in this *static*

approach. Strategies *dynamic* and *stat.dyn* are analog to the other problems. Both strategies are performed with the different separation procedures for the SIs as described in Section 4.1. The exact separation of pair inequalities by inspection is indicated by the suffix *.pairs*, while the suffixes *.DP* and *.exact* refer to the DP-based and exact separation of SIs, respectively.

As test problems, we used the benchmark instances of Fernandes Muritiba *et al.* (2010). The complete set comprises 800 instances with differing characteristics regarding the number of items, the capacity, the item lengths, and the number of conflicts of the items. Similar to BP and VC, we include in our computational analysis only those 169 instances, for which the solution of the RMP took more than one second of computation time in the *standard* CG. We also distinguish two subgroups of instances. Group 1 comprises instances from the classes 1–4 of the benchmark set, while group 2 comprises instances from classes 5–8. The latter are instances for which the item weights are such that no three items i, j, h with $w_h \geq w_i + w_j$ exist, i.e., no SIs with $|S| > 1$ are possible. Furthermore, we report separate results for the hardest instances in terms of computation time. This includes 49 instances for which the solution time of *standard* CG was more than 10 seconds.

Table 5 summarizes the results for BPC. It reveals that a significant improvement can be achieved by using stabilized CG. The biggest gain is already obtained from the *static* use of pair inequalities leading to a speedup of almost factor four. The additional, dynamic separation of violated SIs further decreases computation times resulting in an average speedup of almost factor five over all considered instances. The pure *dynamic* generation of DIs is inferior to the other stabilization strategies. Still, a speedup of approximately factor three compared to *standard* CG can be achieved. For the hard instances, stabilized CG is even more beneficial and leads to an average speedup exceeding factor eight for *stat+dyn.DP*. The results for group 1 are also more in favor for the stabilization strategies than those over all instances. There, average speedups exceed factors between four and five for *static* and *stat+dyn*, respectively. For group 2, speedups reach factors three and four, respectively. Also, Table 5 indicates that for group 2 the separation of pair inequalities and the DP-based heuristic dynamically generate exactly the same number of DIs. This can be explained by the fact that no SIs other than pair inequalities exist for these instances.

Regarding the different separation strategies for the dynamic separation of violated DIs, the separation of pair inequalities by inspection and the separation with the DP-based heuristic perform comparably well. While the former is slightly better averaged over all considered instances as well as for subgroups 1 and 2, the additional effort for the separation in the latter pays off for the hard instances resulting in a slightly better performance. Also, it is obvious from Table 5 that the exact separation of violated SIs is too time consuming for BPC and does not pay off.

As for the other problems, the results regarding the number of iterations are similar to those for the computation times: Stabilized CG results in a significant reduction of the iterations needed for the optimization of the RMP. Moreover, the dynamic addition of SIs yields an additional, considerable gain compared to using DIs in a static fashion only. Table 5 also reveals that the exact separation of violated SIs does not result in a decrease in the number of iterations compared to the other separation strategies. This can be explained by the use of the KP bound (see Section 4.1) in the separation and the potential miss of some SIs with a violation of at least 25% of the most violated SI. Indeed, preliminary tests showed that without using the KP bound the exact separation of violated DIs leads to a slight decrease in the number of iterations relative to the other separation strategies. However, computation times increase disproportionately.

Overall, stabilization with DIs works well for our CG approach to BPC. While speedups can reach a factor of 50, the most successful strategies *static*, *stat+dyn.pairs*, and *stat+dyn.DP* are never slower than *standard* CG. In contrast, the worst performance of these strategies still results in a speedup of almost factor two. Finally, Table 5 indicates that all SIs that we generated in the test problems were DDOIs, since a recovery was never necessary for any of the instances and strategies.

6. Conclusions

Ben Amor *et al.* (2006) introduced DOIs and DDOIs as a general concept to accelerate the CG process, which was then tested for two well structured problems, CS and BP. As they state, the restriction of the

Instances	Algorithm	Solution time (avg./max/min)	# Iterations (avg./max/min)	SP/ RMP	# Dynamic DIs (avg./max/min)	Recovery (# inst./max it.)
All instances ($n = 169$)	*standard*	1.00/1.00/1.00	1.00/1.00/1.00	0.2	–	–/–
	static	0.27/0.58/0.03	0.37/0.73/0.06	0.5	–	–/–
	dyn.pairs	0.35/2.71/0.07	0.18/0.59/0.02	0.2	12281/162521/328	–/–
	dyn.DP	0.35/1.33/0.08	0.16/0.59/0.01	0.3	16715/227607/1317	0/0
	dyn.exact	3.11/12.13/0.48	0.20/0.60/0.05	13.5	2968/15235/494	0/0
	stat+dyn.pairs	0.21/0.63/0.02	0.17/0.55/0.02	0.4	3294/23661/35	–/–
	stat+dyn.DP	0.22/0.54/0.02	0.15/0.55/0.01	0.4	7201/60899/342	0/0
	stat+dyn.exact	2.83/11.25/0.18	0.16/0.58/0.03	22.6	1991/15919/272	0/0
Group 1 ($n = 92$)	*standard*	1.00/1.00/1.00	1.00/1.00/1.00	0.1	–	–/–
	static	0.24/0.48/0.15	0.34/0.64/0.20	0.1	–	–/–
	dyn.pairs	0.35/2.71/0.12	0.13/0.58/0.02	0.1	15741/162521/328	–/–
	dyn.DP	0.33/1.33/0.15	0.08/0.58/0.01	0.1	23886/227607/1317	0/0
	dyn.exact	1.18/6.32/0.48	0.13/0.50/0.060	1.7	3701/15235/494	0/0
	stat+dyn.pairs	0.17/0.63/0.07	0.12/0.53/0.02	0.2	3738/23661/35	–/–
	stat+dyn.DP	0.18/0.53/0.07	0.08/0.47/0.01	0.2	10914/60899/342	0/0
	stat+dyn.exact	0.72/5.60/0.18	0.08/0.50/0.03	3.9	2946/15919/281	0/0
Group 2 ($n = 77$)	*standard*	1.00/1.00/1.00	1.00/1.00/1.00	0.5	–	–/–
	static	0.31/0.58/0.03	0.42/0.73/0.06	0.9	–	–/–
	dyn.pairs	0.34/0.73/0.07	0.25/0.59/0.03	0.4	8147/27207/1517	–/–
	dyn.DP	0.37/0.79/0.08	0.25/0.59/0.03	0.5	8147/27207/1517	0/0
	dyn.exact	5.43/12.13/0.87	0.29/0.60/0.05	27.6	2092/4452/663	0/0
	stat+dyn.pairs	0.25/0.56/0.02	0.24/0.55/0.02	0.6	2763/10439/589	–/–
	stat+dyn.DP	0.26/0.54/0.02	0.24/0.55/0.02	0.7	2763/10439/589	0/0
	stat+dyn.exact	5.35/11.25/0.41	0.25/0.58/0.03	45.1	850/2496/272	0/0
Hard instances ($n = 49$)	*standard*	1.00/1.00/1.00	1.00/1.00/1.00	0.1	–	–/–
	static	0.18/0.38/0.03	0.27/0.56/0.06	0.2	–	–/–
	dyn.pairs	0.31/2.71/0.07	0.07/0.42/0.02	0.1	27629/162521/1824	–/–
	dyn.DP	0.25/1.33/0.08	0.05/0.36/0.01	0.1	37699/227607/2852	0/0
	dyn.exact	1.03/2.97/0.48	0.10/0.29/0.05	8.8	5255/15235/1629	0/0
	stat+dyn.pairs	0.13/0.63/0.02	0.06/0.41/0.02	0.2	6322/23661/975	–/–
	stat+dyn.DP	0.12/0.32/0.02	0.04/0.37/0.01	0.2	15838/60899/1497	0/0
	stat+dyn.exact	0.56/2.71/0.18	0.06/0.27/0.03	11.6	3949/15919/416	0/0

Table 5: Computational results for the bin packing problem with conflicts (BPC)

dual space by DIs leads to less possible intermediate values for the dual multipliers so that a dual-optimal solution is computed faster. In this paper, we extend their results in theory, applications, algorithms, and computational results.

On the theoretical side, we generalize the characterization of a system of DDOIs as stated in Proposition 1. Moreover, instead of deriving properties of dual-optimal solutions directly from the problem under consideration, we introduce several new properties referring to the coefficient matrix of the underlying primal problem. The new comprehensive class of the so-called weighted subset inequalities (WSI) is shown to be DOIs for any problem whose coefficient matrix fulfills the exchange property. A direct consequence is that WSI also hold for VP, and they generalize the subset inequalities prior known to be DOIs for CS. With the help of the row replacement property, the pair inequalities are proven to form a new class of DDOIs for BP.

On the application side, the same pair inequalities are also valid for VC and BPC. For these two problems, DIs have not been tested before (to the best of our knowledge). Furthermore, DOIs and DDOIs can also be applied to problems with general cost-minimization objective, which is only partly covered in the paper at hand (due to the unit-cost assumption made starting from Section 3). In a companion paper (Gschwind and Irnich, 2014), we show that the CG algorithm of Caprara *et al.* (2013) for the temporal knapsack problem benefits from the integration of DOIs. Recently, we discussed and agree with Bianchessi *et al.* (2014) that the split commodities mixed routing problem is an excellent candidate to apply DIs for a vehicle routing problem. We suspect that an observable acceleration of the branch-and-price algorithm can be gained.

We see the most important algorithmic innovation in the *dynamic* generation of DIs. This strategy can either replace or complement the integration of DIs in the form of additional primal columns into the initial

RMP. Several exact and heuristic algorithms for the dynamic generation (= separation) of DIs have been presented. The additional effort for the implementation of separation procedures is small, as shown for CS, BP, and VC. In fact, the most violated WSI in CS and BP are an algorithmic by-product, since they result from intermediate solutions that must anyway be computed when the knapsack-type pricing problem is solved by DP (the currently best known approach in this case). Even some tailored separation heuristics like the ones proposed for BPC are relatively simple to implement.

The possible over-stabilization of the CG process with potential DDOIs is another new idea first coined and analyzed in this paper. The idea goes hand in hand with the dynamic generation of DIs because WSIs form an exponential class of DIs and can therefore not entirely be added to the initial RMP. On the downside, the dynamic addition of non-DDOIs, i.e., DIs that may cut off the entire dual-optimal polyhedron, requires the use of a recovery algorithm as the one presented in Section 4.2. The required iterative application of the recovery algorithm may at the end lead to many additional CG iterations. Fortunately, this never happened in our computational studies. It is indeed reverse, only in rare cases more than one recovery was needed, most of the time none was required at all. Overall, over-stabilization turns out to be advantageous on average for BP, VC, and BPC. For BP, we show that finally much larger speedups can result when DIs are added dynamically.

The extensive computational tests on many known and some additional harder benchmark problems allow some clear statements: For all problems analyzed (CS, BP, VC, and BPC), the newly identified and potential DDOIs together reduce the number of CG iterations when added before proving optimality of the RMP, sometimes significantly. Since the LP-reoptimization of the RMP becomes slightly more time consuming with additional DI columns, the overall reduction in computation time is generally smaller than the reduction in CG iterations. However, for all studied problems the average computation times are reduced compared to state-of-the-art CG algorithms. For the BP example, we compare with an implementation already using constraint aggregation as suggested by Ben Amor *et al.* (2006). Here we are able to confirm their substantial and impressive speedups of factors between two and three for many BP instances. With a mix of statically and dynamically added DIs together with over-stabilization we gain another factor of approx. two for the extensive benchmark set of Sim and Hart (2013) and a factor of 2.5 for the widely used benchmark by Scholl *et al.* (1997). Notably, while for this strategy (*stat+dyn*) the maximum slowdown was by factor 1.3, the maximum speedup was more than factor ten on some instances. Thus, the use of DIs seldom and only gently hampers the CG process, but very often it accelerates. For future applications we expect the dynamic addition of DIs to work best for those problems, in which the solution of the pricing almost completely occupies the overall computation time: any reduction in the number of CG iterations should linearly contribute to a reduction in the solution time.

The results presented in this paper all refer to the solution of the linear relaxation of the extensive formulation. Clearly, the same techniques should also be tested when CG is used in branch-and-price. Such a comparison is, however, a non-trivial task: The overall computation times very much depend on the branching scheme applied, which due to degeneracy of the primal solution may make completely different decisions for the alternative DI strategies, leading to actually incomparable search trees. Even more severe is the fact that some branching decisions are incompatible with some DIs, see, e.g., the branch-and-price algorithm of Alves and Valério de Carvalho (2008) for the multiple length CS. Generic strategies to integrate DIs and branching constitute an interesting avenue for future research.

References

Alves, C. and Valério de Carvalho, J. M. (2008). A stabilized branch-and-price-and-cut algorithm for the multiple length cutting stock problem. *Computers & Operations Research*, **35**(4), 1315–1328.

Alves, C., Valério de Carvalho, J., Clautiaux, F., and Rietz, J. (2014). Multidimensional dual-feasible functions and fast lower bounds for the vector packing problem. *European Journal of Operational Research*, **233**(1), 43–63.

Balas, E. and Padberg, M. W. (1976). Set partitioning: A survey. *SIAM Review*, **18**(4), 710–760.

Balinski, M. (1965). Integer programming: Methods, uses, computation. *Management Science*, **12**(3), 253–313.

Barnhart, C., Johnson, E., Nemhauser, G., Savelsbergh, M., and Vance, P. (1998). Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, **46**(3), 316–329.

Ben Amor, H. and Valério de Carvalho, J. (2005). Cutting stock problems. In Desaulniers *et al.* (2005), chapter 5, pages 131–161.

Ben Amor, H., Desrosiers, J., and Valério de Carvalho, J. M. (2006). Dual-optimal inequalities for stabilized column generation. *Operations Research*, **54**(3), 454–463.

Bettinelli, A., Cacchiani, V., and Malaguti, E. (2014). Bounds and algorithms for the knapsack problem with conflict graph. Technical Report OR-14-16, DEIS – University of Bologna, Bologna, Italy.

Bianchessi, N., Archetti, C., and Speranza, M. G. (2014). An exact solution approach for the split commodities mixed routing problem. Presented at the VeRoLog Conference, Oslo, Norway.

Bode, C. and Irnich, S. (2012). Cut-first branch-and-price-second for the capacitated arc-routing problem. *Operations Research*, **60**(5), 1167–1182.

Caprara, A. and Toth, P. (2001). Lower bounds and algorithms for the 2-dimensional vector packing problem. *Discrete Applied Mathematics*, **111**(3), 231–262.

Caprara, A., Furini, F., and Malaguti, E. (2013). Uncommon Dantzig-Wolfe reformulation for the temporal knapsack problem. *INFORMS Journal on Computing*, **25**(3), 560–571.

Carraghan, R. and Pardalos, P. M. (1990). An exact algorithm for the maximum clique problem. *Operations Research Letters*, **9**(6), 375–382.

Dantzig, G. and Wolfe, P. (1960). Decomposition principle for linear programs. *Operations Research*, **8**, 101–111.

Desaulniers, G., Desrosiers, J., Ioachim, I., Solomon, M., Soumis, F., and Villeneuve, D. (1998). A unified framework for deterministic time constrained vehicle routing and crew scheduling problems. In T. Crainic and G. Laporte, editors, *Fleet Management and Logistics*, chapter 3, pages 57–93. Kluwer Academic Publisher, Boston, Dordrecht, London.

Desaulniers, G., Desrosiers, J., and Solomon, M., editors (2005). *Column Generation*. Springer, New York, NY.

Desrosiers, J., Gauthier, J. B., and Lübbecke, M. E. (2014). Row-reduced column generation for degenerate master problems. *European Journal of Operational Research*, **236**(2), 453–460.

du Merle, O., Villeneuve, D., Desrosiers, J., and Hansen, P. (1999). Stabilized column generation. *Discrete Mathematics*, **194**, 229–237.

Fernandes Muritiba, A. E., Iori, M., Malaguti, E., and Toth, P. (2010). Algorithms for the bin packing problem with conflicts. *INFORMS Journal on Computing*, **22**(3), 401–415.

Fréville, A. (2004). The multidimensional 0-1 knapsack problem: An overview. *European Journal of Operational Research*, **155**(1), 1–21.

Garfinkel, R. S. and Nemhauser, G. L. (1969). The set-partitioning problem: Set covering with equality constraints. *Operations Research*, **17**(5), 848–856.

Gauthier, J. B., Desrosiers, J., and Lübbecke, M. E. (2014). Tools for primal degenerate linear programs. *EURO Journal on Transportation and Logistics*. (In press.).

Gilmore, P. and Gomory, R. (1961). A linear programming approach to the cutting-stock problem. *Operations Research*, **9**, 849–859.

Gilmore, P. and Gomory, R. (1963). A linear programming approach to the cutting stock problem: Part II. *Operations Research*, **11**, 863–888.

Gschwind, T. and Irnich, S. (2014). Solution of temporal knapsack problems by column generation accelerated using dual inequalities. Technical Report LM-2014-05, Chair of Logistics Management, Gutenberg School of Management and Economics, Johannes Gutenberg University Mainz, Mainz, Germany. (In preparation.).

Gualandi, S. and Chiarandini, M. (2014). Graph coloring benchmarks. `https://sites.google.com/site/graphcoloring/home`.

Gualandi, S. and Malucelli, F. (2012). Exact solution of graph coloring problems via constraint programming and column generation. *INFORMS Journal on Computing*, **24**(1), 81–100.

Held, S., Cook, W., and Sewell, E. C. (2012). Maximum-weight stable sets and safe lower bounds for graph coloring. *Mathematical Programming Computation*, **4**(4), 363–381.

Hifi, M. and Michrafy, M. (2007). Reduction strategies and exact algorithms for the disjunctively constrained knapsack problem. *Computers & Operations Research*, **34**(9), 2657–2673.

Hiriart-Urruty, J.-B. and Lemaréchal, C. (1993). *Convex Analysis and Minimization Algorithms, Part 2: Advanced Theory and Bundle Methods*, volume 306 of *Grundlehren der Mathematischen Wissenschaften*. Springer, Berlin, Germany.

Irnich, S. and Desaulniers, G. (2005). Shortest path problems with resource constraints. In Desaulniers *et al.* (2005), chapter 2, pages 33–65.

Kellerer, H., Pferschy, U., and Pisinger, D. (2004). *Knapsack Problems*. Springer, Berlin.

Lee, C. and Park, S. (2011). Chebyshev center based column generation. *Discrete Applied Mathematics*, **159**(18), 2251–2265.

Lübbecke, M. and Desrosiers, J. (2005). Selected topics in column generation. *Operations Research*, **53**(6), 1007–1023.

Malaguti, E., Monaci, M., and Toth, P. (2011). An exact approach for the vertex coloring problem. *Discrete Optimization*, **8**(2), 174–190.

Marsten, R., Hogan, W., and Blankenship, J. (1975). The boxstep method for large-scale optimization. *Operations Research*, **23**, 389–405.

Martello, S. and Toth, P. (1990). *Knapsack Problems: Algorithms and Computer Implementations*. Wiley Intersience Series in Discrete Mathematics and Optimization. Wiley, New York.

Mehrotra, A. and Trick, M. A. (1996). A column generation approach for graph coloring. *INFORMS Journal on Computing*, **8**(4), 344–354.

Östergård, P. R. J. (2002). A fast algorithm for the maximum clique problem. *Discrete Applied Mathematics*, **120**(1-3), 197–207. Special Issue devoted to the 6th Twente Workshop on Graphs and Combinatorial Optimization.

Ozden, M. (1988). A solution procedure for general knapsack problems with a few constraints. *Computers & Operations Research*, **15**(2), 145–155.

Pattillo, J., Youssef, N., and Butenko, S. (2013). On clique relaxation models in network analysis. *European Journal of*

Operational Research, **226**(1), 9–18.
Pferschy, U. and Schauer, J. (2009). The knapsack problem with conflict graphs. *Journal of Graph Algorithms and Applications*, **13**(2), 233–249.
Rousseau, L.-M., Gendreau, M., and Feillet, D. (2007). Interior point stabilization for column generation. *Operations Research Letters*, **35**(5), 660–668.
Sadykov, R. and Vanderbeck, F. (2013). Bin packing with conflicts: A generic branch-and-price algorithm. *INFORMS Journal on Computing*, **25**(2), 244–255.
Scholl, A., Klein, R., and Jürgens, C. (1997). Bison: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Computers & Operations Research*, **24**(7), 627–645.
Sim, K. and Hart, E. (2013). Generating single and multiple cooperative heuristics for the one dimensional bin packing problem using a single node genetic programming island model. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, GECCO '13, pages 1549–1556, New York, NY, USA. ACM.
Valerió de Carvalho, J. M. (1998). Exact solution of cutting stock problems using column generation and branch-and-bound. *International Transactions in Operational Research*, **5**(1), 35–44.
Valério de Carvalho, J. M. (1999). Exact solution of bin-packing problems using column generation and branch-and-bound. *Annals of Operations Research*, **86**, 629–659.
Valério de Carvalho, J. M. (2005). Using extra dual cuts to accelerate column generation. *INFORMS Journal on Computing*, **17**(2), 175–182.
Vance, P., Barnhart, C., Johnson, E., and Nemhauser, G. (1994). Solving binary cutting stock problems by column generation and branch-and-bound. *Computational Optimization and Applications*, **3**, 111–130.
Vanderbeck, F. (1999). Computational study of a column generation algorithm for bin packing and cutting stock problems. *Mathematical Programming*, **86**(3), 565–594.
Vanderbeck, F. (2000). On Dantzig-Wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm. *Operations Research*, **48**(1), 111–128.
Vanderbeck, F. (2005). Implementing mixed integer column generation. In Desaulniers *et al.* (2005), chapter 12, pages 331–358.

Appendix

A. Proofs

Proof of Proposition 1:
(i)⇒(ii) and (ii)⇒(iii): Take a dual-optimal solution π^* from $D^* \cap \{\pi : E^\top \pi \leq e\}$, i.e., π^* is feasible for $\tilde{D}$ (showing (ii)). Since $\tilde{D}$ is a restriction of D, π^* is also optimal for $\tilde{D}$ implying $z_D = z_{\tilde{D}}$.
(iii)⇒(iv): Consequence of LP duality.
(iv)⇒(v): Independent of $(\tilde{\lambda}, y)$, take a primal optimal solution λ^* to P. Then, $(\lambda^*, 0)$ is a feasible for $\tilde{P}$ and has identical objective value. Hence, it is an optimal solution to $\tilde{P}$ because of $z_P = z_{\tilde{P}}$. Therefore, $c^\top \lambda^* = z_{\tilde{P}} \leq c^\top \tilde{\lambda} + e^\top y$.
(v)⇒(vi): Let $(\tilde{\lambda}, y)$ be optimal for $\tilde{P}$. Then there exists a λ^* feasible for P with $c^\top \lambda^* \leq c^\top \tilde{\lambda} + e^\top y$. Also, (λ^*, y^*) with $y^* = 0$ is feasible for $\tilde{P}$ and has identical cost. Thus, (λ^*, y^*) is optimal for $\tilde{P}$ and $Ey^* = 0$.
(vi)⇒(vii) and (vii)⇒(i): Since $Ey^* = 0$, the primal solution $\tilde{\lambda}^*$ is feasible for P. Moreover, y^* is an extreme ray for $\tilde{P}$. The optimality of $(\tilde{\lambda}^*, y^*)$ implies that $\tilde{P}$ is bounded so that $e^\top y^* \geq 0$. Then, $z_{\tilde{P}} = c^\top \tilde{\lambda}^* + e^\top y^* \leq z_P \leq c^\top \tilde{\lambda}^*$. As a consequence, $e^\top y^* = 0$ and $z_{\tilde{P}} = z_P = z_D = z_{\tilde{D}}$. Therefore, every optimal dual solution π^* to $\tilde{D}$ is feasible and herewith optimal for D, i.e., $\pi^* \in D^*$ (showing (vii)) and hence $\pi^* \in D^* \cap \{\pi : E^\top \pi \leq e\}$. □

Proof of Proposition 2:
Assume that the left part (Exch-D) is not fulfilled, i.e.,

$$\sum_{i \in I} (t_i - s_i)\pi_i^* > 0 \tag{2a}$$

We have to show that $\lambda_k^* = 0$ holds for all $k \in J$ with $a_k \geq s$. Consider such a column index $k \in J$: The primal variable λ_k has the associated dual inequality

$$\sum_{i \in I} a_{ik}\pi_i \leq 1. \tag{2b}$$

Since A has the (s,t)-exchange property, we have $a_k - s + t \in A$ with the associated dual inequality

$$\sum_{i\in I}(a_{ik} - s_i + t_i)\pi_i = \sum_{i\in I} a_{ik}\pi_i + \sum_{i\in I}(t_i - s_i)\pi_i \leq 1. \tag{2c}$$

Since π^* is dual feasible, it follows from (2a) and (2c)

$$\sum_{i\in I} a_{ik}\pi_i^* \leq 1 - \sum_{i\in I}(t_i - s_i)\pi_i^* < 1$$

such that the dual inequality (2b) has strictly positive slack. Complementary slackness implies $\lambda_k^* = 0$, i.e., (Exch-P), which completes the proof. □

Proof of Proposition 3:
This is a direct consequence of Proposition 2: The option (Exch-P) is impossible because $\lambda_k^* = 0$ for all $a_k \in A$ with $a_k \geq u_h$, i.e., $a_{hk} \geq 1$ implies $(A\lambda^*)_h = 0 < b_h$. Hence, (Exch-D) must be true implying the above inequality. □

Proof of Theorem 1:
For both CS and VP, it is easy to verify that the matrix A has the (h,t)-exchange property for all $h \in I$, $S \subseteq I$, $t \in \mathbb{Z}_{>0}^S$ with $w_h \geq \sum_{s\in S} t_s w_s$ and $w_{hp} \geq \sum_{s\in S} t_s w_{sp}$ for all $p = 1, 2, \ldots, d$, respectively. Then, the result follows directly from Proposition 3. □

Proof of Remark 1:
(i): We give a constructive proof. If $A\lambda \geq b$ already holds with equality, there is nothing to prove. Otherwise, there exists a row $h \in I$ for which $(A\lambda)_h > b_h$. Choose any $j \in J$ with $\lambda_j > 0$ and $a_{hj} > 0$. Since $a_j \geq u_h$ and A has the $(h,0)$-exchange property, $a_j - u_h \in A$ holds. Thus, let $a_k = a_j - u_h$ be the corresponding column. Then, $\lambda' = \lambda - u_j + u_k$ is another solution with $A\lambda' \geq b$, but with smaller surplus (if any). By repeating this type of exchange procedure, a solution λ' to the system $A\lambda' = b, \lambda' \geq 0$ can be constructed.
(ii): Follows directly from Proposition 3. □

Proof of Proposition 4:
We base our proof on the equivalence of (v)⇒(i) of Proposition 1, i.e., we show (v).

Let $(\tilde{\lambda}, y)$ be a feasible solution to $\tilde{P}$. If $y = 0$ there is nothing to do because $\tilde{\lambda}$ is feasible for P with identical cost and hence optimal also for P.

Otherwise, there is at least one positive component of y corresponding to a valid replacement (h,i). We refer to the respective variable as $y_{(h,i)}$. A replacement cycle $(i_1, i_2, \ldots, i_p, i_1)$ (with the additional definition $i_{p+1} := i_1$) is a cyclic sequence of different row indices such that (i_s, i_{s+1}) is a valid replacement and $y_{(i_s,i_{s+1})} > 0$ for all $s = 1, 2, \ldots, p$. A basic solution to $\tilde{P}$ cannot contain any replacement cycles, since the corresponding columns are linear dependent. Consequently, all valid replacements (h,i) form a *directed acyclic graph* (DAG).

We first show that it is possible to construct, with the help of $(\tilde{\lambda}, y)$, another feasible solution $(\tilde{\lambda}', y')$ to $\tilde{P}$ with identical cost $\mathbf{1}^\top\tilde{\lambda} + e^\top y = \mathbf{1}^\top\tilde{\lambda}' + e^\top y'$ and $\mathbf{1}^\top y' < \mathbf{1}^\top y$. The latter inequality means that we can reduce the infeasibility w.r.t. P. Now, choose a longest path $\mathcal{P} = (h = i_1, i_2, \ldots, i_p = i)$ in the DAG. The maximality of $\mathcal{P}$ implies $(Ey)_h < 0$ and $(Ey)_i > 0$, and, therefore, $(A\tilde{\lambda})_h > (A\tilde{\lambda})_i$. Hence, there exist columns $a_k, k \in J$ with $a_{hk} = 1$ and $a_{ik} = 0$. Let $K \subseteq J$ be the set of these column indices.

All columns $a_k, k \in K$ allow the application of all replacements $(i_1, i_2), \ldots, (i_{p-1}, i_p)$ when these are performed in the following order: Consider a fixed column $a_k, k \in K$. Let $1 \leq q \leq p-1$ be the largest index with $a_{i_q,k} = 1$. Apply the replacements $(i_q, i_{q+1}), (i_{q+1}, i_{q+2}), \ldots, (i_{p-1}, i_p)$. Next, let $1 \leq q' \leq q-1$ be the largest index with $a_{i_{q'},k} = 1$. Apply the replacements $(i_{q'}, i_{q'+1}), (i_{q'+1}, i_{q'+2}), \ldots, (i_{q-1}, i_q)$. Replace q by q' and iterate until $q' = 1$. Note that we used every replacement of $\mathcal{P}$ exactly once.

Due to the row replacement property of A, each replacement of a column $a_k, k \in K$ by $a_k - u_h + u_i$ results in another column $a_{j(k)} \in A$. Iteratively, we perform replacements and update the solution $(\tilde{\lambda}, y)$. In each iteration, we compute $\varepsilon = \min\{y_{i_1,i_2}, y_{i_2,i_3}, \ldots, y_{i_{p-1},i_p}\}$. The replacement of a_k into $a_{j(k)}$ is performed $\mu_k = \min\{\tilde{\lambda}_k, \varepsilon\}$ times. The new feasible solution is $y_{i_s} := y_{i_s} - \varepsilon$ for $s = 1, \ldots, p$, while the remaining components of y are unchanged. Moreover, $\tilde{\lambda}_k := \tilde{\lambda}_k - \mu_k$ and $\tilde{\lambda}_{j(k)} := \tilde{\lambda}_{j(k)} + \mu_k$. Whenever one of the $y_{i_s,i_{s+1}}$ becomes zero, the procedure stops. The new feasible solution $(\tilde{\lambda}', y')$ is the current (updated) solution $(\tilde{\lambda}, y)$. Note that this new solution has identical cost and $\mathbf{1}^\top y' < \mathbf{1}^\top y$ holds.

Also note that it is always possible to perform replacements until one of the $y_{i_s,i_{s+1}}$ is 0 because $\sum_{k\in K} \tilde{\lambda}_k \geq \min\{y_{i_1,i_2}, y_{i_2,i_3}, \ldots, y_{i_{p-1},i_p}\} = \varepsilon'$ holds: The feasibility of $(\tilde{\lambda}, y)$ implies $(A\tilde{\lambda} + Ey)_h = 1 = (A\tilde{\lambda} + Ey)_i$ which together with the maximality if $\mathcal{P}$ means that $(A\tilde{\lambda})_h \geq 1 + y_{i_1,i_2}$ and $(A\tilde{\lambda})_i = 1 - y_{i_{p-1},i_p}$. Therefore, $\sum_{k\in K} \tilde{\lambda}_k \geq \varepsilon'$.

The iterative updates finally result in $y' = 0$ because the replacement procedure eliminates at least one arc is from the DAG for every chosen path longest $\mathcal{P}$. This concludes the proof. □

Proof of Theorem 2:
For all three problems, it is easy to verify that the matrix A has the (h,t)-row replacement property for the respective $h, i \in I$ as specified in the theorem. Then, the results regarding DDOIs follow directly from Proposition 4.

For the statements regarding DOIs, note that the matrix A has the (h,i)-exchange property for the respective pairs $h, i \in I$ for all problems and the results follow from Proposition 3. □

Proof of Proposition 5:
(i) "⇒": Follows by defining $y = a_{i*}\lambda - b_i$ and substituting into the lhs of (1).
"⇐": Follows by multiplication of row i with α and the subsequent addition of the two rows.

(ii) P and $\tilde{P}$ are equivalent to a formulation with the additional column corresponding to the equation $\alpha\pi_h = \pi_i$ (Proposition 1) which is according to (i) equivalent to the aggregated formulation $\tilde{P}'$. Because P and $\tilde{P}'$ are equivalent, so are D and $\tilde{D}'$. Furthermore, it is easy to verify that $z_D = b^\top \pi^* = b'^\top \pi'^*$ holds. Therefore, π'^* is an optimal solution to $\tilde{D}'$. □

Proof of Remark 2:
The aggregation of all rows onto the first row results in a single row $(a'_j)\lambda = b'$ with $a'_j = \sum_{i\in I} a_{ij} w_i / L$, $b' = \sum_{i\in I} b_i w_i / L$, and associated dual value $\pi' = 1$. We show this by induction over the number m of rows that are aggregated onto the first row.

Case $m = 0$: The first row is $(a_{1j})\lambda = b_1$. Using the DDOIs proposed by Ben Amor *et al.* (2006), we can choose the associated dual value as $\pi_1^* = w_1/L$. Multiplication with w_1/L results in $(\frac{a_{1j}w_1}{L})\lambda = \frac{b_1 w_1}{L}$ with associated dual $\pi_1^{*\prime} = 1$.

Case $m-1 \to m$: Given is the first row representing the aggregation of rows 1 to $m-1$ of the form $(\frac{\sum_{i=1}^{m-1} a_{ij} w_i}{L})\lambda = \frac{\sum_{i=1}^{m-1} b_i w_i}{L}$ with $\pi_1^{*\prime} = 1$. Also, row m is $(a_{mj})\lambda = b_m$ with $\pi_m^* = \frac{w_m}{L} = \frac{w_m}{L}\pi_1^{*\prime}$. Aggregating both rows according to Proposition 5 gives

$$(\frac{\sum_{i=1}^{m-1} a_{ij} w_i}{L} + \frac{w_m}{L} a_{mj})\lambda = (\frac{\sum_{i=1}^{m} a_{ij} w_i}{L})\lambda = \frac{\sum_{i=1}^{m-1} b_i w_i}{L} + \frac{w_m}{L} b_m = \frac{\sum_{i=1}^{m} b_i w_i}{L}.$$

The associated dual value is 1.

Moreover, the aggregated entry a_j represents column $j \in J$ of the original model formulation. Because the instance can be solved without any loss, there is at least one column $j \in J$ representing a cutting pattern with no loss, i.e., $\sum_{i\in I} a_{ij} w_i = L$, in the original model. Therefore, the corresponding aggregated entry is 1. Columns with loss result in aggregated entries $a_j < 1$.

Obviously, an optimal solution to the aggregated model is given by choosing $\sum_{i=1}^{m} b_i w_i / L$ times a column $j \in J$ with $a_j = 1$. □

Proof of Proposition 6:
Note first that in set-covering also dominated columns can be eliminated, i.e., a column a_j can be eliminated if there exists another column a_k with $a_j \leq a_k$ (and $c_j \geq c_k$, fulfilled here due to $c_j = c_k = 1$).

Let a_j be the column corresponding to the independent set $S \subseteq I$ with $h \in I$ and $i \notin I$. Because $N(h) \supseteq N(i)$, which implies $i \notin N(h)$ and $h \notin N(i)$, the set $S \cup \{i\}$ is also an independent set and the corresponding column a'_j dominates column a_j. Thus, $a_{hj} > a_{ij}$ cannot hold for an undominated column. As a result, we have $a_{ij} \geq a_{hj}, \forall j \in J$ after eliminating all dominated columns and, hence, row i is dominated by row h and can be eliminated. □

B. Dynamic Programming-based Separation of Subset Inequalities for the Bin Packing Problem with Conflicts (BPC)

We refer to Section 4.1 in which we claimed that the heuristic DP-based separation algorithm for identifying violated SIs can be implemented to run in $\mathcal{O}(mL)$ time. Recall that in the BPC an item $i \in I$ has an associated interval $\mathcal{I}_i := (a_i, b_i)$. Any two items $i, j \in I$ are in conflict (indicated by $\{i, j\} \in \mathcal{E}$) if and only if $\mathcal{I}_i \cap \mathcal{I}_j \neq \emptyset$.

In a preprocessing step, we loop over all items $h \in I$ and compute $S_h := \{i \in I : N(h) \cup \{h\} \supseteq N(i), w_h \geq w_i\}$. Moreover, the smallest interval $\mathcal{I}_{S_h}$ containing $\bigcup_{i \in S_h} (a_i, b_i)$ can be determined by computing the left- and rightmost interval borders of the items in S_h, i.e., $a_{S_h} := \min_{j \in S_h} a_j$ and $b_{S_h} := \max_{j \in S_h} b_j$ and setting $\mathcal{I}_{S_h} = (a_{S_h}, b_{S_h})$. This step requires $\mathcal{O}(m)$ time per item, overall $\mathcal{O}(m^2)$ time.

The actual separation procedure works as follows: Loop over all states $w = 1, 2, \ldots, L$ at the final stage m of the DP. Each state represents a solution, which is an independent set S_w with weight w. The determination of S_w takes $\mathcal{O}(m)$ time. Now determine the smallest interval $\mathcal{I}_{S_w}$ covering $\bigcup_{j \in S_w} \mathcal{I}_j$. This can be done, similar as before, by computing $a_{S_w} := \min_{j \in S_w} a_j$ and $b_{S_w} := \max_{j \in S_w} b_j$ and setting $\mathcal{I}_{S_w} = (a_{S_w}, b_{S_w})$. Also, this step requires only $\mathcal{O}(m)$ time. An additional loop over all $h \in I$ now allows the direct $\mathcal{O}(1)$ test, whether or not h and S_w together form a possible SI. Indeed, $w_h \geq w$ is trivial and $S_h \supseteq S_w$ is checked via $a_{S_h} \geq a_{S_w}$ and $b_{S_h} \leq b_{S_w}$. In the positive case, the violation of the DI $(h \leftarrow S_w)$ is $\sum_{s \in S_w} \bar{\pi}_s - \bar{\pi}_h$. Summing up, the outer loop requires $\mathcal{O}(L)$ time, while all steps inside that loop require $\mathcal{O}(m)$ time, yielding the $\mathcal{O}(mL)$ result.

C. Detailed Computational Results for CS

Table 6 presents results for CS by subclass.

Instances	Algorithm	Solution time (avg./max/min)	# Iterations (avg./max/min)	SP/ RMP	# Dynamic DIs (avg./max/min)
All instances ($n = 240$)	*standard*	1.50/8.73/0.93	1.77/29.00/1.31	17.6	–
	static	1.00/1.00/1.00	1.00/1.00/1.00	13.7	–
	dynamic	1.03/4.80/0.44	0.86/9.00/0.41	6.5	5590/18100/247
	stat+dyn	0.83/1.53/0.29	0.75/1.00/0.31	9.8	1120/3742/0
$L =$ 1.500.000 ($n = 120$)	*standard*	1.49/7.06/0.98	1.70/13.00/1.32	18.6	–
	static	1.00/1.00/1.00	1.00/1.00/1.00	14.4	–
	dynamic	0.97/4.55/0.49	0.85/7.00/0.41	6.6	5567/18100/247
	stat+dyn	0.82/1.11/0.29	0.75/1.00/0.31	10.5	1131/3742/0
$L =$ 500.000 ($n = 120$)	*standard*	1.50/8.73/0.93	1.85/29.00/1.31	9.7	–
	static	1.00/1.00/1.00	1.00/1.00/1.00	8.1	–
	dynamic	1.08/4.80/0.44	0.87/9.00/0.60	3.6	5614/17362/483
	stat+dyn	0.84/1.53/0.29	0.74/1.00/0.35	5.8	1108/3544/0
with small items ($n = 120$)	*standard*	1.33/7.06/0.93	1.54/13.00/1.31	21.8	–
	static	1.00/1.00/1.00	1.00/1.00/1.00	17.3	–
	dynamic	0.99/4.55/0.58	0.86/7.00/0.59	7.9	6711/18100/247
	stat+dyn	0.87/1.45/0.55	0.77/1.00/0.58	12.1	1538/3742/0
w/o small items ($n = 120$)	*standard*	1.66/8.73/0.99	2.01/29.00/1.51	9.6	–
	static	1.00/1.00/1.00	1.00/1.00/1.00	7.7	–
	dynamic	1.06/4.80/0.44	0.86/9.00/0.41	3.4	4470/11906/483
	stat+dyn	0.79/1.53/0.29	0.72/1.00/0.31	5.8	701/1731/0

Table 6: Detailed computational results for the cutting stock problem (CS)