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

Automatic Dantzig-Wolfe Reformulation of Mixed Integer Programs

Martin Bergner · Alberto Caprara · Alberto Ceselli · Fabio Furini · Marco E. Lübbecke · Enrico Malaguti · Emiliano Traversi

September 20, 2011 · revised September 20, 2012; November 1, 2013

Abstract Dantzig-Wolfe decomposition (or reformulation) is well-known to provide strong dual bounds for specially structured mixed integer programs (MIPs). However, the method is not implemented in any state-of-the-art MIP solver as it is considered to require structural problem knowledge and tailoring to this structure. We provide a computational proof-of-concept that the reformulation can be automated. That is, we perform a rigorous experimental study, which results in identifying a score to estimate the quality of a decomposition: after building a set of potentially good candidates, we exploit such a score to detect which decomposition might be useful for Dantzig-Wolfe reformulation of a MIP. We experiment with general instances from MIPLIB2003 and MIPLIB2010 for which a decomposition method would not be the first choice, and demonstrate that strong dual bounds can be obtained from the automatically reformulated model using column generation. Our findings support the idea that Dantzig-Wolfe reformulation may

A preliminary version of this paper appeared in [3].

Martin Bergner was supported by the German Research Foundation (DFG) as part of the Priority Program "Algorithm Engineering" under grants no. LU770/4-1 and 4-2.

Martin Bergner · Marco E. Lübbecke
Operations Research, RWTH Aachen University, Kackertstraße 7, D-52072 Aachen, Germany, E-mail: {martin.bergner, marco.luebbecke}@rwth-aachen.de

Alberto Ceselli
Dipartimento di Informatica, Università degli Studi di Milano, Via Bramante 65, I-26013 Crema, Italy, E-mail: alberto.ceselli@unimi.it

Fabio Furini
LAMSADE, Université Paris-Dauphine, Place du Maréchal de Lattre de Tassigny, F-75775 Paris, France, E-mail: fabio.furini@dauphine.fr

Enrico Malaguti
DEI, Università di Bologna, Viale Risorgimento 2, I-40136 Bologna, Italy, E-mail: enrico.malaguti@unibo.it

Emiliano Traversi
LIPN, Équipe AOC, Université Paris 13, 99 Avenue Jean-Baptiste Clément, F-93430 Villetaneuse, France, E-mail: emiliano.traversi@lipn.univ-paris13.fr

hold more promise as a general-purpose tool than previously acknowledged by the research community.

Keywords Dantzig-Wolfe decomposition · column generation · block-diagonal matrix · matrix re-ordering · automatic reformulation · hypergraph partitioning

Mathematics Subject Classification (2000) 90C11 · 49M27 · 65K05

1 Introduction

Dantzig-Wolfe reformulation (DWR) of mixed integer programs (MIPs) is a computationally very successful approach to produce high-quality solutions for well-structured discrete optimization problems like vehicle routing, crew scheduling, cutting stock, p-median, generalized assignment, and many others [28]. The common structure of these problems is the bordered block-diagonal form of the constraint matrix of the MIP formulation, which reflects that otherwise independent subproblems are coupled only by some linking constraints. This structure usually gives rise to a column generation based solution approach.

It is generally agreed that DWR needs tailoring to the application at hand and is quite far from being a general-purpose tool: It is the *user* who does not only know *that* there is an exploitable structure present but also *what* it looks like, and *how* to exploit it algorithmically. In particular, in view of the automatic use of general-purpose cutting planes in all modern MIP solvers, this is unsatisfactory.

1.1 Our Contribution

In this paper we give a computational proof-of-concept that the DWR process can be automated and applied to a general MIP even when the latter seemingly does not expose the matrix structure for which DWR is classically applied. We perform a suite of experiments, the results of which can be taken as advice on what empirically constitutes a good (or bad) DWR. Remarkably, a key ingredient—re-arranging a matrix into bordered block-angular form—has been available for a long time. Also automatically applying DWR to a *given* structure is a concept that is implemented in several frameworks. However, these two components have not been combined in a MIP context before. In this paper, we provide the missing link by proposing how to re-arrange a matrix into a structure that is experimentally *well-suited* for DWR, in the sense that a subsequent column generation approach consistently closes a significant portion of the integrality gap. Our main contributions are summarized as follows:

- We reveal that the constraint matrix of a general MIP can be re-arranged for DWR in many different ways, necessitating a way to *a priori* evaluate the quality of a re-arrangement;
- we perform a rigorous experimental study which results in a proxy measure for this quality, which we call the *relative border area*;
- besides the classical bordered block-diagonal matrix structure, also a *double*-bordered block-diagonal (also called *arrowhead*) matrix is amenable to DWR when applying a variant of Lagrangian decomposition. The re-arrangement of

matrices into both forms *having a good border area* can be accomplished via a hypergraph partitioning algorithm;
- for a set of medium sized instances from MIPLIB2003 and MIPLIB2010 our reformulations on average lead to comparable or stronger root node dual bounds w.r.t. a state-of-the-art MIP solver with default parameter settings (i.e., cutting planes, preprocessing, etc. enabled);
- on a non-negligible fraction of these instances, we could automatically identify reformulations yielding computational performances that are globally better than those of state-of-the-art MIP solvers;
- our computational success is based on the observation that the constraint matrix of a MIP may not originally contain a Dantzig-Wolfe decomposable form, but can be "forced" into such a form in almost all cases;
- *performance variability* has raised great attention in the computational MIP literature lately; we show experimentally that this phenomenon is very pronounced in the context of DWR.

As we re-arrange matrices with the goal of applying a DWR, we will use the notions of re-arrangement of a matrix and (Dantzig-Wolfe) decomposition interchangeably, as the former immediately leads to the latter in our context.

1.2 Related Literature

For a general background on DWR of MIPs, column generation, and branch-and-price we refer to the recent survey [28] and the primer [9] in the book [8] that also devotes several chapters to the most important applications.

There are several frameworks which perform DWR of a general MIP, and handle the resulting column generation subproblems in a generic way such as `BaPCod` [27], `DIP` [23], `G12` [22], and `GCG` [16]. In all cases, the bordered block-diagonal matrix structure needs to be *known and given* to the algorithm by the user. In [16] it is shown that such a generic reformulation algorithm performs well on bin packing, graph coloring, and p-median problems. Tebboth, in his thesis [26], derives some decomposable matrix structures of a *linear* program (LP) when it is formulated in a specific modeling language. A similar approach of indicating the matrix structure via key words in the modeling language is taken in [7, 13, 22, 25] among others. All proposals have in common that the user, in one way or another, needs to make available *her knowledge* about the decomposition to be applied.

Specially structured matrices, like bordered block-diagonal forms, play an important role in several fields, e.g., in numerical linear algebra. Therefore, several proposals exist to re-arrange the rows and columns of a matrix in order to reveal such forms. A typical motivation is to prepare a matrix for parallel computation, like for solving linear equation systems, see, for instance, [2] and the many references therein. The goal usually is to identify a given number of independent blocks (of almost equal size) with as few constraints in the border as possible (see below for more formal statements). Some works like [30] mention the possible use of such re-arrangements in DWR of LPs, but we know of no actual experiment with MIPs. The method in [12], for speeding up interior point methods, is based on graph partitioning as is ours. An exact branch-and-cut algorithm for detecting a bordered block-angular structure was proposed in [4].

Attempts to evaluate the re-arranged matrices in terms of suitability for DWR were done rudimentarily in the LP case only [26]. We are not aware of any attempts to evaluate the quality of a decomposition in terms of suitability of DWR for the MIP case (which has an undoubtedly larger potential). In fact, it is not even known what characterizes a "good" decomposition in this context, and our paper gives computationally supported guidance in this respect.

2 Partial Convexification and Dantzig-Wolfe Reformulations

A sketch of DWR applied to a MIP is as follows (see e.g., [9] for details). Consider a general MIP

$$\max\{c^t x : Ax \leq b,\ Dx \leq e,\ x \in \mathbb{Z}^{n-r} \times \mathbb{Q}^r\}\ . \tag{1}$$

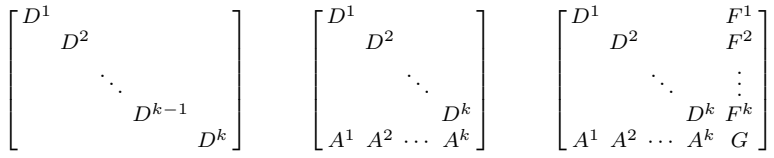
Let $P := \{x \in \mathbb{Q}^n : Dx \leq e\}$. The polyhedron $P_{IP} := \text{conv}\{P \cap \mathbb{Z}^{n-r} \times \mathbb{Q}^r\}$ is called the *integer hull* w.r.t. constraints $Dx \leq e$. In a DWR based on a single block of constraints $Dx \leq e$ we express $x \in P_{IP}$ as a convex combination of the (finitely many) extreme points Q and (finitely many) extreme rays R of P_{IP}; this leads to

$$\begin{aligned}
&\max\ c^t x && (2)\\
&\text{s.t. } Ax \leq b\\
&\quad x = \sum_{q \in Q} \lambda_q q + \sum_{r \in R} \mu_r r\\
&\quad \sum_{q \in Q} \lambda_q = 1\\
&\quad \lambda \in \mathbb{Q}_+^{|Q|},\ \mu \in \mathbb{Q}_+^{|R|}\\
&\quad x \in \mathbb{Z}^{n-r} \times \mathbb{Q}^r && (3)
\end{aligned}$$

where each $q \in Q$ and $r \in R$ represent vectors encoding extreme points and rays, respectively, and variables λ and μ correspond to weights in their combination.

It is well-known that the resulting LP relaxation is potentially stronger than that of (1) when $P_{IP} \subsetneq P$ [17], in which case the *dual bound* one obtains from (2)–(3) is stronger than the one from (1). This is a main motivation for performing the reformulation in the first place. This *partial convexification* w.r.t. the constraints $Dx \leq e$ corresponds to implicitly replacing P with P_{IP} in (1). This can be done, in principle, by explicitly adding *all* valid inequalities for P_{IP} to (1). When this is impracticable, the implicit description is in a sense the best one can hope for.

The reformulated MIP (2)–(3) contains the *master constraints* $Ax \leq b$, the convexity constraint, and the constraints linking the *original* x variables to the *extended* λ and μ variables. In general, MIP (2)–(3) has an exponential number of λ and μ variables, so its LP relaxation needs to be solved by column generation. The *pricing subproblem* to check whether there are variables with positive reduced cost to be added to the current *master LP* calls for optimizing a linear objective function over P_{IP}, so it is again a MIP. The effectiveness of the overall approach hinges crucially on our ability to solve this subproblem, either by a general-purpose solver or by a tailored algorithm to exploit its specific structure, if known.



(a) block-diagonal **(b)** bordered block-diagonal **(c)** double-bordered block-diagonal

Fig. 1 Matrix forms

DWR is usually applied when matrix D has a *block-diagonal* form as depicted in Fig. 1(a) where $D^i \in \mathbb{Q}^{m_i \times n_i}$ for $i = 1, \ldots, k$ (and D is 0 elsewhere). In this case, k disjoint sets of constraints are identified: $Dx \leq e$ partitions into $D^i x^i \leq e^i$, $i = 1, \ldots, k$, where $x = (x^1, x^2, \ldots, x^k)$, with x^i being an n_i-vector for $i = 1, \ldots, k$. Every $D^i x^i \leq e^i$ individually is then convexified in the above spirit. We denote by k the *number of blocks* of the reformulation. A matrix of the form as in Fig. 1(b) with $A^i \in \mathbb{Q}^{m_\ell \times n_i}$, $i = 1, \ldots, k$ is called *(single-)bordered block-diagonal*. For general MIPs, often enough, constraints are not separable by variable sets as above, and a *double-bordered block-diagonal* (or *arrowhead*) form is the most specific structure we can hope for, i.e., the constraint matrix of (1) looks like Fig. 1(c) with $F^i \in \mathbb{Q}^{m_i \times n_\ell}$, $i = 1, \ldots, k$, and $G \in \mathbb{Q}^{m_\ell \times n_\ell}$. The m_ℓ constraints associated with the rows of A^i are called the *coupling* (or *linking*) *constraints* and the n_ℓ variables associated with the columns of F^i are called the *linking variables*, denoted by x^ℓ. We intuitively speak of the k *blocks* D^i, $i = 1, \ldots, k$, and the remaining *border*. Each of the k groups of constraints $D^i x^i + F^i x^\ell \leq e^i$ can be convexified. Let $P^i := \{x \in \mathbb{Q}^{n_i + n_\ell} : D^i x^i + F^i x^\ell \leq e^i\}$. Denote by P^i_{IP} the associated integer hull and, respectively, by Q^i and R^i the set of extreme points and rays of P^i_{IP}. Then, the resulting DWR reads

$$\max\ c^t x \tag{4}$$

$$\text{s.t.} \sum_{i=1}^{k} A^i x^i + G x^\ell \leq b$$

$$(x^i, x^\ell) = \sum_{q \in Q^i} \lambda^i_q q + \sum_{r \in R^i} \mu^i_r r \qquad (\forall i) \tag{5}$$

$$\sum_{q \in Q^i} \lambda^i_q = 1 \qquad (\forall i)$$

$$\lambda^i \in \mathbb{Q}_+^{|Q^i|},\ \mu^i \in \mathbb{Q}_+^{|R^i|} \qquad (\forall i)$$

$$x \in \mathbb{Z}^{n-r} \times \mathbb{Q}^r \tag{6}$$

where each $q \in Q$ and $r \in R$ is augmented with x^ℓ components. This formulation generalizes the *explicit master format* [21] handling both, the presence of linking variables and a generic number of blocks. It is a variant of Lagrangian decomposition, similarly used in [12,26]. Instead of replacing each linking variable with a copy for each block it appears in, and adding constraints that ensure consistency of these copies, this coordination is taken care of by the original x variables. Furthermore, by keeping all x^i variables in the master, one can enable several features of

general purpose MIP solvers, like the separation of generic cuts, advanced branching techniques, preprocessing, etc., without any additional implementation issues. We use the above formulation in our experiments. We can further obtain a master problem formulation by projecting out the original variables, and introducing for each linking variable a set of constraints which ensures consistency in different blocks. However, we remark that the techniques and findings described in the rest of the paper are not strictly dependent on the particular master problem format.

3 Automatic Detection of an Arrowhead Form

Potentially, every MIP model is amenable to DWR, even if its structure is not known in advance (from the modeler or from other sources). In the latter case, we need to detect a structure algorithmically. We need hence to decide: *i*) which constraints of the MIP (if any) to keep in the master problem; *ii*) the number of blocks k and *iii*) how to assign the remaining constraints to the different blocks. In other words, we need to partition the set of the original constraints into one subset representing the master and several subsets representing the blocks.

To this end, we follow the ideas proposed in [2] and [12] which build on well-known connections between matrices and hypergraphs. Precisely, once the number k of blocks is fixed, we propose a procedure for obtaining a decomposition consisting of two main ingredients: *i*) a systematic way to produce a hypergraph starting from the constraint matrix, and *ii*) an algorithm that partitions its vertex set into k subsets (blocks). The partition obtained corresponds to a re-arrangement of the constraint matrix of the original problem in such a way that it presents an arrowhead form ready to be used as input for the model (4)-(6).

3.1 Hypergraphs for Structure Detection

We use two different kinds of hypergraphs defined in the following which are the input of the partitioning algorithm.
Row-Net Hypergraph. Given the matrix A, we construct a hypergraph $H = (V, R)$ as follows. Each column j of matrix A defines a vertex $v_j \in V$, and each row i of matrix A defines a hyperedge $r_i \in R$ linking those vertices $v_j \in V$ whose corresponding variable j has non-zero entry $a_{ij} \neq 0$ in the row i.
Row-Column-Net Hypergraph. Given the matrix A, we construct a hypergraph $H = (V, R \cup C)$ as follows. Each entry $a_{ij} \neq 0$ of matrix A defines a vertex $v_{ij} \in V$. For every row i of A, we introduce a hyperedge $r_i \in R$ which spans all vertices $v_{ij} \in V$ with $a_{ij} \neq 0$; analogously we introduce a hyperedge $c_j \in C$ for every column j spanning all vertices for which $a_{ij} \neq 0$.

3.2 Hypergraph Partitioning

In order to obtain a decomposition we heuristically solve a minimum weight balanced k-partition problem on one of the two hypergraphs. This problem is to

partition the vertices of a hypergraph V into k components V_i $(i = 1, \ldots, k)$ such that the sum of weights on hyperedges containing vertices in more than one V_i is minimized. The partition is said to be *balanced* if $|V_i| \approx n/k$.

For a given k-partition of a hypergraph, the blocks are obtained by grouping together constraints corresponding to hyperedges $r_i \in R$ consisting of vertices v belonging to the same component; in the case of the row-net hypergraph, hyperedges r spanning different components correspond to linking constraints and are hence kept explicitly in the master problem. Similar, for a row-column-net hypergraph, the hyperedges c spanning different components correspond to linking variables. We recall that by construction, a partition on a row-net hypergraph does not allow partitions with linking variables.

The idea behind both constructions is to obtain "homogeneous" blocks, i.e., blocks consisting of constraints on similar sets of variables and limiting at the same time the number of linking constraints (in both cases) and linking variables (when a row-column-net hypergraph is used). To take into account slightly unbalanced partitions, dummy nodes that are not incident to any hyperedge are included in each graph. The possibility of assigning weights to hyperedges enables us to penalize different components of the border differently; for example linking variables may be less desirable when they are discrete, or linking variables may be more or less difficult to handle in the subsequent column generation process than linking constraints, etc.

3.3 A Very Special Case: The Temporal Knapsack Problem

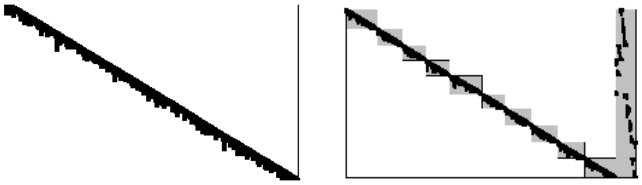
The arrowhead structure of a matrix is so general that it cannot happen that there is no such structure to detect. Therefore, "detection" can also be understood as "forcing" a matrix into a particular structure. We illustrate this on the *temporal knapsack problem* [5], also known as *unsplittable flow on a line* or *resource allocation*.

The problem is defined as follows. There are n items, the i-th of which has size w_i, a profit p_i, and is *active* only during a time interval $[s_i, t_i)$. A subset of items has to be packed into a knapsack of capacity C such that the total profit is maximized and the knapsack capacity is not exceeded at any point in time. It suffices to impose that the capacity constraint is satisfied at the discrete points in time s_i, $i = 1, \ldots, n$. Let $S_j := \{i : s_i \leq j \text{ and } j < t_i\}$ denote the set of active tasks at time j, and x_i a binary variable equal to 1 if task i is selected, a binary program for the problem reads:

$$\max \Big\{ \sum_{i=1}^{n} p_i x_i : \sum_{i \in S_j} w_i x_i \leq C,\ \forall j \in \{s_1, \ldots, s_n\}, x_i \in \{0,1\},\ \forall i \in \{1, \ldots, n\} \Big\}. \quad (7)$$

Fig. 2(a) shows that the coefficient matrix of this formulation does not contain any non-trivial blocks (i.e., the entire matrix is one block), although all non-zero entries in a column are associated with consecutive rows.

Applying the procedure explained in the previous subsection based on a row-column-net hypergraph produces a decomposition with only a few linking variables, see Fig. 2(b), and "reveals" a Dantzig-Wolfe decomposable form. This form



(a) original matrix **(b)** "detected" arrowhead form

Fig. 2 constraint matrix of an instance of the temporal knapsack problem, (a) original and (b) "forced" into arrowhead form with $k = 10$ blocks.

immediately suggests how to convexify groups of constraints in the spirit of (2)–(3) to (much) better describe the integer hull implicitly. A DWR of the second matrix does not only produce a strong relaxation but also allows to quickly solve instances to optimality by branch-and-price, significantly outperforming a state-of-the-art MIP solver on the standard formulation (7) [5].

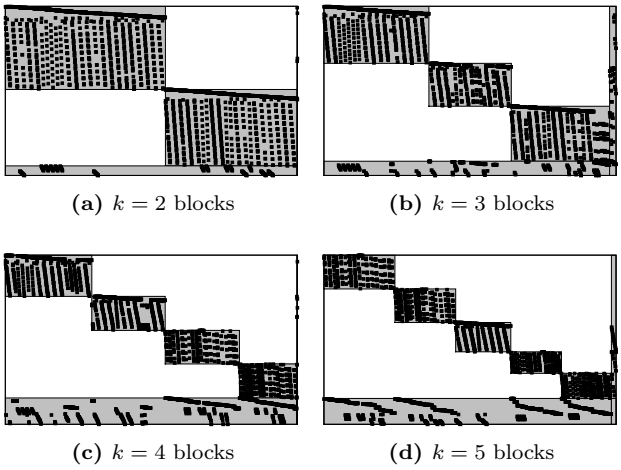
4 Experimenting with a Generic Decomposition Framework

An important lesson we learned from preliminary experiments [3] is that we are not looking for "the" decomposition of a matrix. There is a much larger degree of freedom than originally expected: an arrowhead form is not unique; changing the input parameters of our re-arrangement algorithm can very much vary the results, and most importantly, seemingly small changes in the resulting decomposition may lead to very different behavior in the subsequent column generation process (both, in terms of dual bound quality and running time). Such sensitivity is a well-documented phenomenon in MIP solving in general [20]. Figs. 3 and 4 show for instance the influence of the number k of blocks and the choice of the weights on hyperedges, respectively, on the *visual* shape of the resulting arrowhead form. Not least, this non-uniqueness of decompositions immediately calls for *a priori* evaluation criteria for a given decomposition w.r.t. its usefulness for DWR. In this study, we concentrate on approaching this question experimentally.

In the following we describe the *benchmark instances* used in our computational experiments, the specific setting used in our *experimental framework* (i.e., model (4)–(6) of Sect. 2) and the *sets of decompositions* achieved using the procedure described in Sect. 3 using three different settings.

4.1 Benchmark Instances

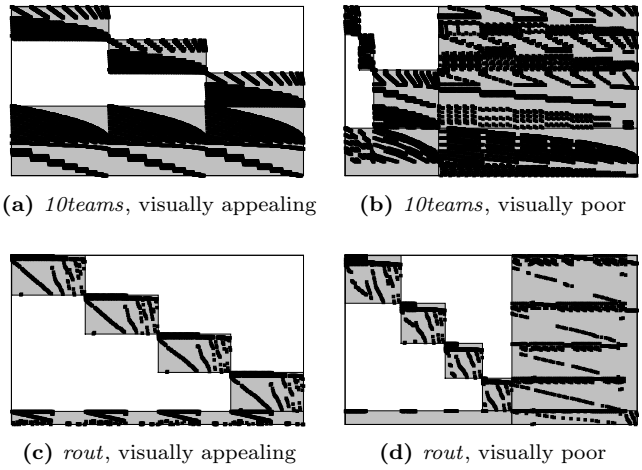
We did not experiment on instances with known structure: in these cases the effectiveness of DWR has already been proven, and indeed, the purpose of our study is to prove that DWR can work for general MIPs. Instead, in order to confirm the generality of the proposed method we tested our algorithm on MIPLIB2003 [1] and MIPLIB2010 [20] instances. We selected a subset of instances, for which (a) the density is between 0.05% and 5%, (b) the number of non-zeros is not larger than 20,000, and (c) the fraction of discrete variables is at least 20%. The rationale



(a) $k = 2$ blocks **(b)** $k = 3$ blocks

(c) $k = 4$ blocks **(d)** $k = 5$ blocks

Fig. 3 Influence of the number k of blocks on the shape of the arrowhead form (*fiber*); a smaller k tends to give a smaller border.



(a) *10teams*, visually appealing **(b)** *10teams*, visually poor

(c) *rout*, visually appealing **(d)** *rout*, visually poor

Fig. 4 Visual differences for different decompositions (resulting from different settings in the partitioning procedure). Top row (a,b) shows instance *10teams* with $k = 3$ blocks; bottom row (c,d) shows instance *rout* with $k = 4$ blocks.

behind this choice is the following: (a) if the instance is not sparse enough, no useful arrowhead form can be detected, and therefore it is easy to tell *a priori* that DWR is not a promising option; (b) for large instances the partitioning heuristics may fail in finding good partitions, and therefore the potential of our approach cannot be estimated; and (c) without a sufficient degree of integrality no gains in the dual bound are possible from a decomposition.

4.2 Experimental Framework

All experiments to assess the performances of model (4)-(6) are carried out on an Intel Core^{TM} i7-870 PC (2.93 GHz, 8MB cache, 8GB memory) running Linux 2.6.34 (single thread). In all our experiments the CPU time limit is set to 1 hour.

For solving the column generation MIP subproblems we use the branch-and-cut algorithm implemented in `CPLEX` 12.2, with single thread and default parameter settings. In order to obtain a generic multiple pricing strategy, each time a new incumbent solution having negative reduced cost is found during the optimization of a MIP subproblem, the corresponding column is inserted into the master. Whenever a MIP subproblem is found to be unbounded, its integrality conditions are relaxed, and an extreme ray is generated and added to the master by solving the resulting LP problem to optimality.

For solving the master problem LPs we use the simplex algorithm of `CPLEX` 12.2, with default parameter settings, again single thread. We implemented the dual variable stabilization method described in [11]. We keep as a stability center the dual solution giving the best Lagrangian bound so far, and we enforce a penalty factor ϵ whenever each dual variable takes values outside an interval of width δ around the stability center. At each column generation iteration we change ϵ, independently for each dual variable, to a value randomly chosen between 0 and 10^{-4}. Whenever the stability center changes we set $\delta := 0.00005 \cdot |z - v|$, uniformly for all dual variables, where z is the current master problem LP value, and v is the best Lagrangian bound found so far. An upper bound of 50.0 is imposed on δ in any case. The stabilization is activated on problems with more than 400 constraints when, during the column generation process, the gap $|z - v|$ is between 1% and 80% of the current $|z|$ value. We experimented with different parameter settings, but these values gave best performance; we experimentally observed that, on our datasets, this stabilization mechanism is enough to overcome potential convergence problems. As we did not go for efficiency, no further performance improvement method was implemented.

4.3 Sets of Decompositions

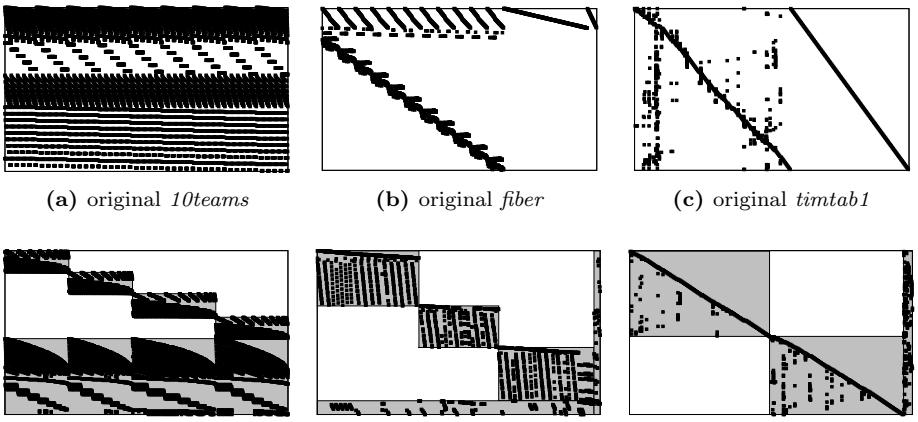
Different decompositions can be obtained using the hypergraphs described in Sect. 3.1 with different sets of input parameters. For instance, completely different decompositions can be derived by changing the number k of blocks, the number of dummy vertices, and the weights of the hyperedges given as input parameters to the partitioning algorithm.

We do not optimally solve the NP-hard minimum weight balanced k-partition problem. Instead, we use the heuristic multilevel hypergraph partitioning algorithm in [18], of which the package `hMETIS` [19] is an implementation. In particular, the `hMETIS` heuristics follow a multilevel recursive bisection paradigm, working in three phases: coarsening, partitioning, and uncoarsening. Coarsening aims at constructing a sequence of successively smaller hypergraphs by contracting hyperedges with a so-called "FirstChoice" scheme; then, balanced partitioning is performed

on the contracted hypergraph by bisection; finally, during uncoarsening, the contracted hypergraph is expanded by successive projections, following backward the coarsening steps, and running after each projection a local refinement procedure (FM), that is based on tabu search. See [18] for details. `hMETIS` can solve the partitioning problem needed in our procedure in a few seconds. All `hMETIS` parameters were kept at their default values, except the random seed that was set to 1.

Decompositions without Linking Variables. As first set, we generate decompositions leading to the standard single bordered structure with no linking variables. This shape is important because the majority of formulations used in practice for DWR present this shape. We create a row-net hypergraph imposing a weight of 1 on hyperedges corresponding to constraints. We add a number of dummy vertices equal to 20% of the number of non-zero entries in the constraint matrix. We set $k = 2, \ldots, 20$ for the number of blocks, obtaining 19 decompositions for each instance. An example of a matrix in a MIP model, and the corresponding detected structure are reported in Figs. 5(a) and 5(d), respectively.

Balanced Decompositions. In the most general setting we search for decompositions in which both constraints and variables can be part of the border. We create a row-column-net hypergraph with a weight of 1 on hyperedges corresponding to continuous variables, a weight of 2 on hyperedges corresponding to integer variables, and a weight of 5 on hyperedges corresponding to constraints. We add a number of dummy vertices equal to 20% of the number of non-zero entries in the constraint matrix. We set $k = 2, \ldots, 10$ for the number of blocks, obtaining 9 decompositions for each instance. Figs. 5(b) and 5(e) show an example matrix as given in a MIP model, and the structure detected using this approach, respectively.



(a) original *10teams* **(b)** original *fiber* **(c)** original *timtab1*

(d) *10teams*, detected struct. **(e)** *fiber*, detected structure **(f)** *timtab1*, detected struct.

Fig. 5 Patterns of non-zero entries (black dots) in the coefficient matrix of a MIPs from the MIPLIB2003 [1]; the areas with grey background in (d)–(f) emphasize the embedded structure; (a)–(c) show that matrix structure directly from the MPS file, with a re-ordering of rows and columns according to a decomposition found with (d) no linking variables, (e) balanced settings, (f) no linking constraints.

Decompositions with few Linking Constraints. We created a row-column-net hypergraph imposing a weight of 1 on hyperedges corresponding to continuous variables, a weight of 2 on hyperedges corresponding to integer variables and a weight of 10^5 on hyperedges corresponding to constraints. We added a number of dummy vertices equal to 20% of the number of non-zero entries in the constraint matrix. We set $k = 2, \ldots, 10$ for the number of blocks, obtaining 9 decompositions for each instance. See Figs. 5(c) and 5(f) for an example of a MIP model matrix and the structure detected in this way, respectively.

Altogether, we obtain a benchmark sample of around 1500 decompositions of 39 instances, which cover a wide range of different possible shapes. These decomposition are used for testing our computational framework in the following section.

5 Linking Input and Output Measures

In the previous sections, we explained how the set of decompositions has been created and solved via DWR. We now offer a complete analysis of the results obtained by considering the link between the following output and input measures:

Output Measures. The definition of "usefulness" of a given decomposition is already an open issue. To obtain as much methodological insight as possible, we primarily measure such a usefulness in terms of the root node dual bound of the obtained master relaxation, being a reliable indicator on the effectiveness of the corresponding convexification process.

Efficiency is not the primary aim of our investigation. Yet, in order to evaluate a decomposition's computational potential, we consider as a second usefulness measure the computing time needed to complete the column generation process at the root node (including the time needed to obtain the decomposition, which is negligible). We intuitively call this the computation time of a decomposition.

Input Measures. In an effort to understand which parameters affect most the output measures, we take into account the following four *direct indicators*: number of blocks, percentage of linking variables, percentage of linking constraints, and the percentage of border area. The procedure explained in Sect. 3 can produce decompositions with different values of the direct indicators, this can be done by explicitly fixing the number of blocks a priori or by properly changing the objective function of the graph partitioning problem. As second set of indicators we consider the average density of the blocks and the average integrality gap of the subproblems. These are called *indirect indicators* because it is not possible with our procedure to obtain decompositions minimizing or maximizing them.

Methodology of the Analysis. In each of the following subsections we consider each of these input measures independently, we indicate the expected behavior from theory and we present our experimental results, highlighting an eventual correlation to output measures.

In particular, each subsection synthesizes the results of our experiments in two figures, having the following structure. Each dot corresponds to one of the decompositions generated by the partitioning algorithm. The values on the horizontal (vertical) axis correspond to a particular input (output) measure.

The integrality gap closed by a decomposition is computed by $1 - |OPT - DWR|/|OPT - LP|$, where OPT is the value of a best integer solution found by `CPLEX` in one hour of computing time, DWR is the dual bound found by column generation using a particular decomposition, and LP is the LP relaxation value of the corresponding instance. On a few decompositions column generation could not be completed within the given time limit; in these cases DWR was set to the best Langrangian relaxation value found during column generation. We normalized the computing time t of every decomposition of each instance h as $(t - \mu_h)/\sigma_h$ with μ_h being the mean running time and σ_h being the standard deviation of running time of all decompositions created for that specific instance. For each correlated input-output pair, we fit a `LOESS` curve [6] with a span size of $s = 0.75$.

We further plot the different types of decompositions with different colors, where the decompositions with no linking variables are plotted with a black dot (•), balanced decompositions with a dark grey dot (•) and decompositions with few linking constraints with a light grey dot (•). For every figure in the sequel, we provide a figure disaggregated by instance in the appendix.

In the end of the analysis we propose a proxy measure, suitable for guiding an *a priori* choice among several potential decompositions. This proxy measure is used in Sect. 6, where more computationally related benchmarks are discussed and where we compare the performances of our decompositions with `CPLEX`.

5.1 Direct Indicators

As sketched above, direct indicators are those that can be either chosen as input parameters or optimized in the hypergraph partitioning algorithm.

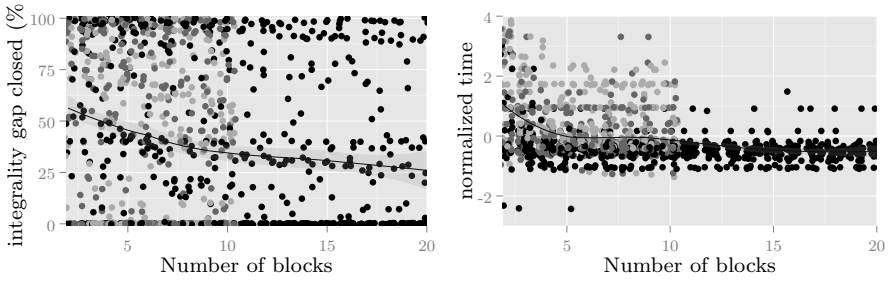
5.1.1 Number of Blocks

We first observe the influence of the number k of blocks, that is chosen as a parameter in our partitioning algorithms, on the output measures.

Expectation 1 *As the number of blocks increases, both the integrality gap closed at the root node and the computing time decrease.*

In fact, on one hand a higher number of blocks means more (smaller) subproblems, that can be optimized faster than a few large ones. On the other hand, more blocks means a more fragmented convexification, that might lead to looser bounds. We also remark that, even if we expect the master relaxation to improve as k decreases, there is no guarantee that a simple reduction in k is useful in terms of bound quality. In fact, different values of k give different decompositions, in which different regions are convexified, and no *a priori* dominance can be established between them. The results of our experiments are synthesized in Fig. 6.

Observation 1 *The average computing time decreases as the number of blocks increases.*



(a) Gap **(b)** Time

Fig. 6 Influence of the number k of blocks on the output measures.

Observation 2 *Decompositions without linking variables usually have smallest computation times among all decompositions per instance.*

Observation 3 *When the number of blocks is small, it is more likely to close a high integrality gap.*

This can be observed by looking at the distribution of the dots in Fig. 6, and becomes more evident by looking at the instance-by-instance plots reported in the appendix. Keeping the number of blocks relatively small is a key point if we want to keep the advantage of a strong dual bound. From the experiments it is hence also clear that we should focus on decompositions with no more than 5 or 6 blocks. On the other hand, it is important to notice that decompositions including 20 blocks exist, in which 100% of the integrality gap is closed at the root node.

This partially confirms our theoritical expectations: few blocks yield tight bounds, but as their number increases both good and bad decompositions exist, thus motivating the search for measures ranking good ones.

5.1.2 Percentage of Linking Variables

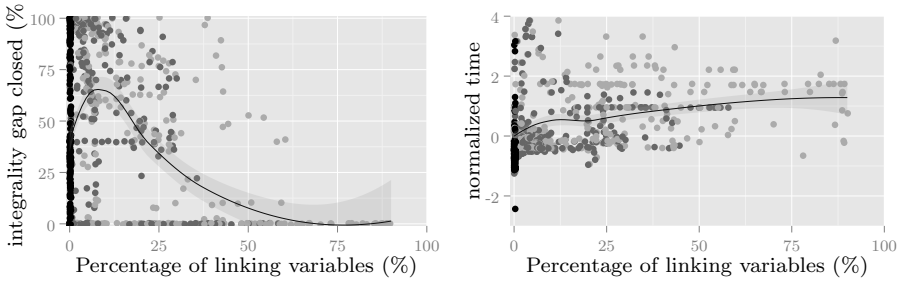
Second, we consider the fraction of linking variables $\frac{n_\ell}{n}$ in each decomposition (see Fig. 7).

Expectation 2 *As the fraction of variables in the border increases, the computing time increases.*

In fact, including variables in the border is not reducing the number of variables in the subproblem, but is only allowing different copies of the same variable to take different values in the subproblems. In a column generation setting, then, different subproblems may blindly generate solutions which are found to be incompatible at a master stage, thus slowing down the convergence of the process.

Expectation 3 *As the fraction of variables in the border increases, the integrality gap closed decreases.*

As recalled in Sec 2, including variables in the border is similar to including constraints: due to the analogy with Lagrangian decomposition, more variables in the border lower the potential gain by the reformulation.



(a) Gap **(b)** Time

Fig. 7 Influence of the percentage of linking variables on the output measures.

Observation 4 *The fraction of variables in the border and the integrality gap closed are linked by weak negative correlation.*

In fact, as the fraction of variables in the border increases, the computing time increases, and therefore many instances run into time limit, so the integrality gap closed decreases. Hence, our expectations are confirmed. An opposite trend can be observed for very low fractions of variables in the border. Indeed, good decompositions exist for some instances, that rely on including linking variables into the border. For these instances, the single border decompositions generated by our partitioning algorithms may be substantially worse than those with a few linking variables, thereby worsening the average gap closed.

Observation 5 *Computing times tend to get worse as the fraction of variables in the border increases.*

The expected unstable behavior is observed also in practice.

5.1.3 Percentage of Linking Constraints

Third, in Fig. 8 we take into account the fraction of linking constraints $\frac{m_\ell}{m}$ in each decomposition.

Expectation 4 *As the fraction of constraints in the border increases, the integrality gap closed at the root node decreases.*

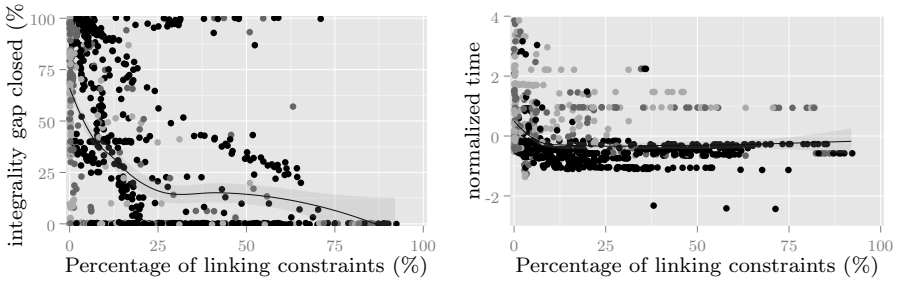
The rationale is the following: when a constraint is included in a block, its corresponding region of feasibility is convexified, possibly yielding tighter bounds.

At the same time, in general subproblems are MIPs, and therefore it is hard to tell *a priori* the effect on computing times obtained by adding constraints.

Observation 6 *As the fraction of constraints in the border increases, the integrality gap closed at the root node decreases.*

This matches our theoretical expectation.

Observation 7 *Computing time is higher for very low percentages of constraints in the border.*



(a) Gap **(b)** Time

Fig. 8 Influence of the percentage of linking constraints on the output measures.

Such an observation is not fully matching a modeler intuition, as when the number of linking constraints is zero, we may have a pure block-diagonal form, which performs perfectly in a decomposition setting. In an attempt to explain the experimental phenomenon, we consider two factors. First, the decompositions with very low percentages of linking constraints include those with largest percentage of linking variables, that according to Obs. 5 yield higher computing time. Second, allowing a relevant percentage of constraints to stay in the border means having more freedom in the composition of blocks. In other words, it facilitates the partitioning algorithm to fit the remaining constraints in the best block. This explanation meets also with practice: in several partitioning problems, for instance, a natural decomposition is obtained only by assigning a relevant fraction of the constraints to the border.

In contrast to our results in Sect. 5.1.2, where having very few linking variables does not guarantee to provide tight bounds, having very few linking constraints does. That is, adjusting the fraction of linking constraints experimentally offers more control on the integrality gap closed with respect to adjusting the fraction of linking variables.

5.1.4 Percentage of Border Area

Fourth, we investigate the effect of adjusting linking variables and linking constraints simultaneously, by considering the following *relative border area*

$$\beta = \frac{m_\ell \cdot n + m \cdot n_\ell - m_\ell \cdot n_\ell}{m \cdot n} \quad . \tag{8}$$

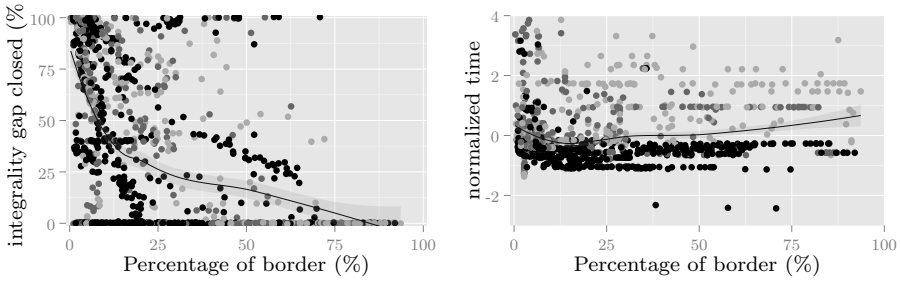
This is the ratio between the border "area" and the entire matrix "area," and is a way of encoding the intuition on visually appealing decompositions sketched in Sect. 4: having a small border area is what a human expert usually aims to, while trying to make a block diagonal structure appear by selectively choosing border elements. Trivially, for a fixed number of blocks, a MIP whose constraint matrix is in pure block diagonal form can be decomposed with border area zero.

Expectation 5 *Decompositions having a reduced set of constraints and variables in the border tend to yield tighter dual bounds.*

The rationale comes directly from Exp. 3 and 4: each constraint that appears in the border corresponds to a constraint that is not convexified; in the same way, each column that appears in the border requires duplicated variables and linking constraints to be created and relaxed in a Lagrangian decomposition fashion.

From the theoretical point of view, it is hard to predict the relation between relative border area and computing times. However, following a modeler intuition, two extreme cases can be considered: when all the constraints and variables are in the border, the problem becomes an LP, that can be optimized efficiently; on the opposite, when no constraints nor variables are in the border, we are facing the original MIP. Therefore, such an observation suggests to investigate on the following with the results of our experiments summarized in Fig. 9:

Expectation 6 *Decompositions having a reduced set of constraints and variables in the border tend to yield higher computing times.*



(a) Gap **(b)** Time

Fig. 9 Influence of the relative border area on the output measures.

Observation 8 *A lower relative border area correlates with a better dual bound.*

As can be observed, the trend in Fig. 9 is more clear with respect to that of results on both percentage of linking constraints (Fig. 8) and linking variables (Fig. 7).

Observation 9 *Tests whose computing time is high, often correspond to decompositions having relative border area either considerably low or considerably high.*

The higher percentage of slow runs with low relative border area meets Exp. 6, and that of slow runs with considerably high relative border area meets Exp. 2 and matches Obs. 5. Furthermore, fewer linking constraints implies more constraints in the subproblems, and hence potentially more extreme subproblem solutions to generate via column generation, thus yielding higher computing time.

5.2 Indirect Indicators

We also consider *indirect* indicators, that is, measures that cannot be reflected in the objective function of our heuristic minimum k-way hypergraph partitioning approach. Therefore, they are not optimized directly *during the construction of a decomposition*, but only assessed *a posteriori*.

5.2.1 Average Density of the Blocks

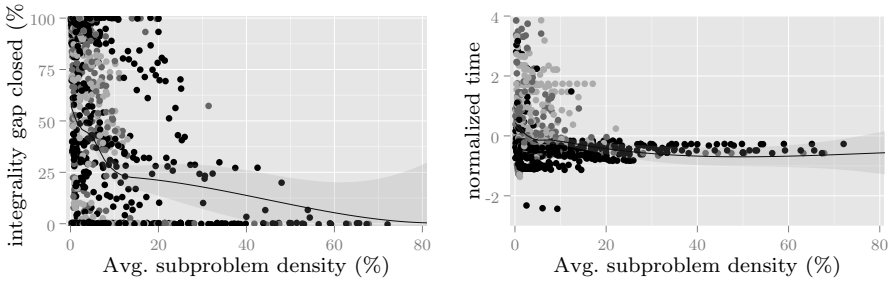
First, we investigate the influence of the density of subproblems, computed as

$$\delta_i = \frac{nz_i}{n_i \cdot m_i} \quad \text{and} \quad \delta = \frac{1}{k}\sum_{i=1}^{k} \delta_i \ ,$$

where nz_i denotes the number of non-zero coefficients in D^i. Each decomposition is evaluated according to the average value δ over its blocks.

Expectation 7 *Dense subproblems are more difficult to solve, thus yielding higher computing time.*

This is often observed in the behaviour of MIP solvers. Our computational results are summarized in Fig. 10.



(a) Gap **(b)** Time

Fig. 10 Influence of the average subproblem density on the output measures.

Observation 10 *As the average subproblem density increases, the integrality gap closed tends to decrease.*

Observation 11 *For low values of average subproblem density, as the average subproblem density increases, the computing time tends to decrease, while no particular trend is observed as the average subproblem density further increases.*

In order to explain this phenomenon we compared the average subproblem density of each decomposition in our experiments with the corresponding relative border area, see Fig. 11. We observe a strong correlation, and in fact Obs. 8 and 10 are coherent; moreover, the correlation in case of Obs. 9 and 11 is reversed, i.e., lower density decompositions close a large fraction of the integrality gap but take longer to compute whereas decompositions with a lower relative border area both tend to have good dual bounds and a lower computation time.

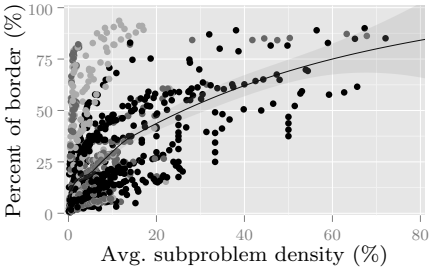


Fig. 11 Correlation between relative border area and average subproblem density.

5.2.2 Average Integrality Gap of Subproblems

Finally, we analyze the relation between average integrality gap of the subproblems and output measures. In order to obtain such an analysis, we considered the MIP subproblems of each block of each decomposition independently, we set its objective function coefficients to those in the original objective function of the problem, we measured its LP relaxation value (LP), and then let `CPLEX` run until proven optimality (that is, in the vast majority of our experiments) or until a time limit of 60 seconds was reached (value ILP). The integrality gap was estimated as $|(ILP - LP)/ILP|$, and for each decomposition the average among its subproblems was computed, excluding those subproblems for which `CPLEX` could not find any feasible solution within the time limit, and those having zero ILP value.

We stress that such a measure, besides being indirect in the sense discussed at the beginning of the section, requires a potentially non-trivial MIP computation for each subproblem, and is therefore interesting for a methodological insight only.

Theory suggests that

Expectation 8 *A larger integrality gap in the subproblems may imply a larger potential for a stronger master dual bound.*

This expectation has also been mentioned in the context of Lagrangean relaxation [17]. Trivially, a subproblem integrality gap of zero for *whatever* objective function indicates that no improvement of the dual bound is possible compared to the original LP relaxation. For the computing time, we expect the following

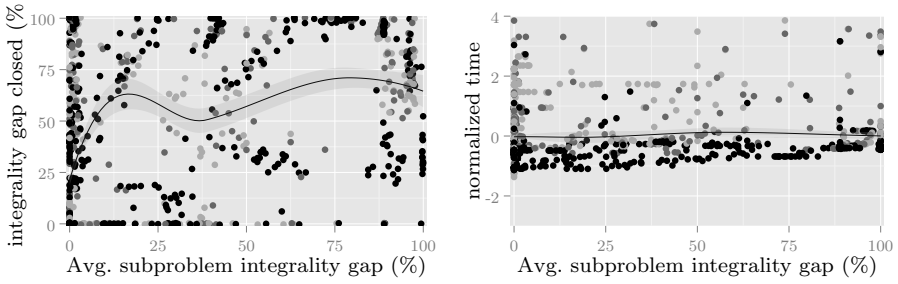
Expectation 9 *The computing time increases as the average integrality gap in the subproblems increases.*

This is due to the fact that closing the integrality gap is the main task of any MIP solver in tackling each subproblem. Our results are reported in Fig. 12.

Decompositions corresponding to points on the horizontal axis are actually those tests hitting time limits. In our experiments we found the following.

Observation 12 *Both integrality gap closed and computing time correlate weakly to average subproblem integrality gap.*

Future experiments may reveal the usefulness of a measure related to the subproblem integrality gap, perhaps by considering a more suitable definition of the objective function.



(a) Gap **(b)** Time

Fig. 12 Influence of the average integrality gap at the root node of the subproblems on the output measures.

5.3 Evaluating and Exploiting a Proxy Measure

We remark that, when evaluating a decomposition, a strong dual bound cannot constitute the ultimate quality measure, as this can be trivially maximized by reformulating the *whole* original problem into a single subproblem (which amounts to optimally solving the original MIP in the subproblem in a single shot).

From the analysis reported in the previous sections we drew the conclusion that relative border area is the most appropriate proxy measure to estimate the quality of a decomposition. In particular, it is the only measure exhibiting three appealing features at the same time. First, the measure ranges from zero (best) to one (worst), being zero for block-angular matrices. Second, it correlates negatively with integrality gap closed. Third, it is eventually positively correlated also with the time needed to compute the bound of the decompositions. In order to be more detailed on this third feature, we present a further analysis on our results: we rank the decompositions by relative border area, and we consider four classes of decompositions, corresponding to the four quartiles in such a ranking; in Table 1 we report the average output measures for each quartile: the table is composed of four rows, one of each quartile, and four columns, reporting quartile, range of values corresponding to the quartile, average normalized integrality gap closed, and average normalized computing time over decompositions in each quartile.

We come to the following

Observation 13 *A better tradeoff between integrality gap closed and computing time can be obtained by looking for decompositions with low relative border area.*

In fact, as reported in Table 1, decompositions having relative border area in the fourth quartile are on average worse than those in all other quartiles in terms of both computing time and integrality gap closed.

For the remaining quartiles, decompositions having low relative border area tend to provide higher integrality gap closed at the price of higher computing time. Indeed, this matches Exp. 5 and 6, and supports the general intuition that additional computational effort is needed to obtain better bounds.

In Table 2 we report, for each MIPLIB instance, the details of the DWR with minimum relative border area among all those generated by our graph partitioning

Quartile	range	norm. int. gap closed	norm. cpu time
1	(0.0031,0.0546]	71.97	0.10
2	(0.0546,0.1380]	47.46	−0.07
3	(0.1380,0.3290]	29.42	−0.15
4	(0.3290,0.9360]	12.80	0.11

Table 1 Average output measures for relative border area quartiles

algorithms, as described in Sect. 4.3. Listed are: the instance features (name, number of rows and columns), number of blocks, linking variables, and linking constraints, maximum number of variables and constraints in a block, number of LP problems solved during column generation and corresponding CPU time, number of MIP subproblems solved during the column generation process and corresponding CPU time, number of columns and extreme rays generated during the optimization process. The table is split in two horizontal blocks, corresponding to MIPLIB 2003 and MIPLIB 2010 instances, respectively. As can be seen, no obvious correlation appears between the prediction given by the proxy measure and the single features of the chosen decomposition; therefore our proxy measure proves to be unbiased with respect to the other decomposition parameters.

Observation 14 *Among those generated by our graph partitioning algorithms, a decomposition corresponding to the best measure has either linking constraints or linking variables, but not both, in the vast majority (36 out of 39) of the instances.*

We remark that this phenomenon is not simply induced by definition (8), as a low relative border area can also be achieved by including in the border very few variables and constraints at the same time. That is, our proxy measure seems to match an intuitive behavior reported in the literature: for each instance, that is the representative of a potentially difficult combinatorial optimization problem, Lagrangian relaxation may be more promising than Lagrangian decomposition, or vice versa. Still, in a few cases, combining the two techniques may be promising.

Observation 15 *Among those generated by our graph partitioning algorithms, a decomposition corresponding to the best measure has no linking variables in the majority (26 out of 39) of the instances.*

This confirms another important modeler's intuition: DWR unfolds its potential when relaxing a few linking constraints is enough to decompose the problem in more tractable subproblems.

6 Comparison to a State-of-the-Art MIP Solver

Finally, we sketch an answer to a very fundamental computational question: are there generic MIP instances for which it is worthwhile to perform DWR with an automatically generated decomposition instead of applying traditional cutting-planes methods? The overall analysis presented in Sect. 5 motivates us to propose the following two "automatic" algorithms.

instance features			best decomposition features										
				linking		maximal		LP		MIP		generated	
instance	cols	rows	blocks	var	cons	var	cons	#	time (s)	#	time (s)	col	rays
10teams	2025	230	9	0	95	225	15	53	0.53	468	1.69	365	0
aflow30a	842	479	2	0	28	430	230	548	40.82	1094	52.66	2097	0
aflow40b	2728	1442	10	0	39	348	179	359	55.59	3580	14.89	3490	0
fiber	1298	363	2	0	22	713	178	1168	11.7	2334	528.77	4658	0
fixnet6	878	478	2	0	14	436	235	899	2664.52	1796	173.92	3721	0
gesa2	1224	1392	2	26	0	625	696	75	0.53	148	17.14	860	0
gesa2-o	1224	1248	2	0	26	611	611	462	7.12	922	106	3191	0
glass4	322	396	2	12	0	178	212	20	0.01	37	2657.16	256	1
harp2	2993	112	10	0	39	369	9	86	0.24	850	1.34	793	0
manna81	3321	6480	2	77	0	1854	3546	896	300.23	1790	234.59	2962	0
mkc	5325	3411	2	0	31	2911	1815	201	21.43	400	84.78	1467	0
modglob	422	291	2	0	8	211	144	1385	2727.36	2768	876.17	17595	0
noswot	128	182	5	0	9	26	35	5	0	20	1.26	31	0
opt1217	769	64	4	0	16	192	12	31	0.01	120	0.06	120	0
p2756	2756	755	3	0	16	918	257	405	32.86	1212	202.09	5048	0
pp08a	240	136	8	0	8	30	16	31	0.01	160	0.46	254	10
pp08aCUTS	240	246	8	0	8	30	32	34	0	168	0.58	224	12
rout	556	291	5	0	16	111	55	20	0	95	5.9	319	0
set1ch	712	492	20	0	12	35	24	14	0.02	260	0.59	364	0
timtab1	397	171	2	13	0	206	86	22	0.03	42	244.56	320	0
timtab2	675	294	2	17	0	348	148	10	0.03	20	3271.48	228	0
tr12-30	1080	750	2	12	0	552	375	2	0	3	0.43	32	0
vpm2	378	234	2	7	0	196	117	19	0.01	36	1.04	140	0
beasleyC3	2500	1750	2	0	26	1252	862	309	2358.21	616	1242.35	5219	0
csched010	1758	351	2	1	55	823	171	102	0.81	202	1040.41	953	0
enlight13	338	169	2	25	0	196	92	35	0.05	68	1068.67	190	0
gmu-35-40	1205	424	2	0	9	432	217	18	0.04	26	0.08	30	4
m100n500k4r1	500	100	2	351	0	440	54	542	38.48	1082	48.62	2393	0
macrophage	2260	3164	2	7	0	1203	1702	8	0.03	16	3547.05	92	0
mcsched	1747	2107	5	0	32	349	415	45	0.06	220	84.24	780	0
mine-166-5	830	8429	2	0	596	416	3928	214	4.97	426	1421.13	2278	0
mine-90-10	900	6270	2	0	92	450	3090	23	0.03	44	2983.29	316	0
neos-686190	3660	3664	3	59	118	1324	1265	82	75.33	244	3330.84	1548	0
pigeon-10	490	931	2	96	40	274	477	183	18.26	364	25.85	1769	0
pw-myciel4	1059	8164	2	0	165	530	4009	59	0.12	114	27.38	146	1
ran16x16	512	288	16	0	16	32	17	25	0.01	384	0.92	475	0
reblock67	670	2523	2	0	69	335	1228	21	0.06	40	529.66	299	0
rmine6	1096	7078	2	0	354	557	3439	35	0.21	68	836.68	331	0
rococoC10-001000	3117	1293	8	0	82	447	407	160	3.36	1272	10.37	875	0

Table 2 Details of the DWR of minimum proxy measure (8), among those generated by our graph partitioning algorithms, for the corresponding instance

DWR auto: for each instance, consider all the decompositions generated by the graph partitioning algorithms as described in Sect. 4.3, pick one minimizing the relative border area (8), keep the original variables in the master, including the possible linking ones (as explained in Sect. 2), and perform a DWR of the resulting blocks; then solve the resulting master problem by column generation.

DWR best: for each instance, consider all the decompositions generated by the graph partitioning algorithms as described in Sect. 4.3, perform column generation on each of the resulting master problems, and keep a decomposition giving the best dual bound.

That is, the algorithm "DWR auto" provides a benchmark for assessing the potential of using the border area for finding decompositions yielding tight dual bounds, whereas "DWR best" aims at assessing the potential of the overall automatic DWR mechanism. Note that "DWR best" is extremely time consuming and only meant as an assessment of the method's potential.

As a benchmark MIP solver we use `CPLEX` 12.2 with full default settings, in particular with default preprocessing and cutting planes enabled. As for DWR, the `CPLEX` integrality gap closed is computed by $1-|OPT-CPLEX|/|OPT-LP|$, where OPT is the value of the best integer solution found by `CPLEX` in one hour of computing time, $CPLEX$ is the dual bound found by `CPLEX` in the root node,

after preprocessing and cut generation, and LP is the LP relaxation value of the corresponding instance.

6.1 Quality of the Dual Bound

The first set of experiments aims at assessing the quality of the bound that can be obtained using the DWR approach. Our goal is to show that by using such an algorithm, it is often possible to obtain a better description of the convex hull of the full problem than that obtained by `CPLEX` through a cutting plane process.

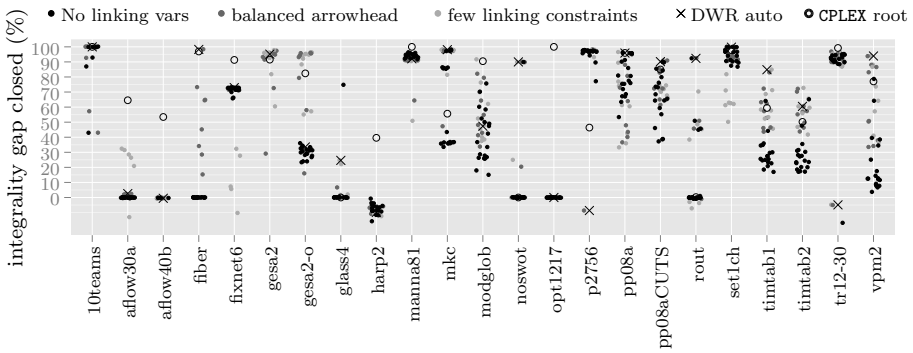


Fig. 13 Distribution of fraction of integrality gap closed for all decompositions (MIPLIB2003)

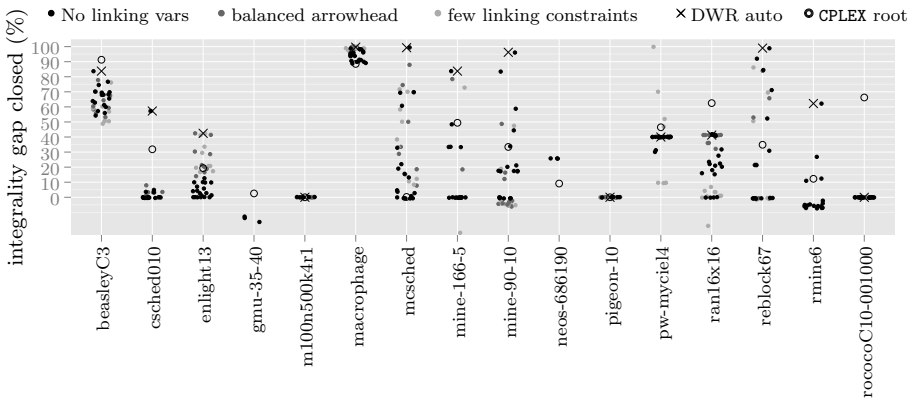


Fig. 14 Distribution of fraction of integrality gap closed for all decompositions (MIPLIB2010)

Figs. 13 and 14 depict the distribution of fraction of integrality gap closed for all the about 1500 decompositions (see Subsection 4.3) we tried for MIPLIB2003

and MIPLIB2010 instances, respectively. Every dot represents one decomposition. On two tests (instances *tr12-30* and *neos-686190*), column generation could not be completed within the time limit, and therefore the gap of the best Lagrangian bound (potentially worse than the LP relaxation bound), obtained during the column generation iterations, is reported [17]. One can see that a decomposition with the respective best proxy measure (marked by a $\times$) most of the time leads to a very good choice among all decompositions we tried with respect to the root node integrality gap closed, also compared to the bound obtained by `CPLEX` after the cutting plane process at the root node (marked by a $\bigcirc$).

In Table 3 we detail, for each instance in the dataset, our results for both the "DWR auto" and "DWR best" methods, compared to the results obtained by `CPLEX` when stopped after the cutting plane process at the root node as well as the time `CPLEX` needs to solve the original problem to optimality.

Listed are: the instance name, the value of an optimal solution or best solution computed by `CPLEX` in 1 hour (opt*), the value of the LP relaxation (LP) and the corresponding duality gap (LP gap). Then, three vertical blocks follow, corresponding to "DWR auto," "DWR best," and `CPLEX`, respectively. For each we report the integrality gap, the improvement with respect to the LP relaxation bound (a value of zero is reported if the decomposition is not able to improve over the LP relaxation bound), the number of nodes used to attain that bound, and the time needed to compute that bound.

The table is composed of two horizontal blocks, that refer to the 23 selected instances of MIPLIB2003 and to the 16 selected instances of MIPLIB2010, respectively. The last row of each block reports average values.

On 16 out of the 23 MIPLIB 2003 and 11 out of the 16 MIPLIB 2010 instances, the dual bound found by our DWR approach improves on `CPLEX`'s root node bound with default settings, and in four more instances the bound is the same.

On the average, whereas "DWR best" clearly outperforms `CPLEX`, "DWR auto" is still competitive, experimentally supporting the meaningfulness of our proxy measure for the quality of a decomposition.

6.2 Overall Performance Comparison

Since DWR and `CPLEX` produce bounds of different quality with different computational efforts, in a second set of experiments we aim at comparing the trade-off between computing time and quality of the dual bound given by the two methods. This is an overall index designed to measure the potential of both methods in actually solving MIPs to optimality, and to help answering the ultimate question of whether good dual bounds can be provided by DWR in reasonable time

We remark that, apart from the computation of bounds, the remaining machinery in branch-and-price and branch-and-cut algorithms is equivalent, as similar preprocessing, heuristics, constraint propagation, branching techniques, etc., can be implemented. Also the re-optimization process in the nodes of the branch-and-bound tree is similar, provided the same decomposition is kept along the search tree. In fact, state of the art branch-and-price frameworks [16] keep pools of columns, and therefore the optimization of a node is likely to start from a restricted master problem containing at least the columns forming an optimal basis

for the father node, as in standard branch-and-bound codes, and very often many more high quality columns already generated during the optimization of siblings.

At the same time, re-implementing the techniques included in a state-of-the-art branch-and-cut solver was beyond of the scope of this paper. Therefore, in order to perform a comparison, we decided to consider the DWR giving best proxy measure in "DWR auto," and to take as a performance index the ratio between the time required by our algorithm to obtain the dual bound at the root node, when using such a decomposition, and the time required by `CPLEX` with default settings to obtain, either at the root node or through branching, the same bound. In the comparison we included also "DWR best."

The results are also reported in Table 3. The last vertical block for `CPLEX` with default settings shows the number of nodes and the CPU time needed to reach the same bound as "DWR auto;" the ratio between "DWR auto" time and `CPLEX` time for obtaining such a bound. Below each vertical block we report the fraction of instances for which "DWR auto" was faster than `CPLEX` (< 1), within one order of magnitude w.r.t. `CPLEX` (< 10) and within two orders of magnitude ($< 10^2$).

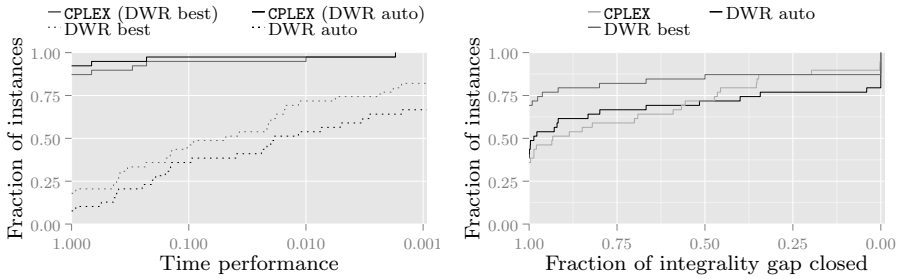
On average, `CPLEX` is much faster in solving the root node relaxation. At the same time, on many instances in which good decompositions can be found, `CPLEX` needs to explore many branching nodes, and to spend a high CPU time for matching the bound given by "DWR auto." Finally, it is interesting to note that in more than half of the instances (up to 52.17% of MIPLIB2003 and up to 56.25% of MIPLIB2010), "DWR auto" is still within a factor 10^2 from `CPLEX`, making us optimistic that, after a suitable software engineering process, our approach would become computationally competitive on more instances.

6.3 Performance Profiles

Finally, we compared "DWR auto" and "DWR best" to `CPLEX` using the methodology of performance profiles [10]. We performed two different analyses, the first one is based on time performance and the second one is based on bound performance, disregarding the time needed to compute it.

The time analysis is displayed in Fig. 15(a), where we show two different comparisons, the first one concerns "DWR auto" and `CPLEX` (black lines, dashed and solid), the second one concerns instead "DWR best" and `CPLEX` (grey lines, dashed and solid). For each pair of algorithms and for each instance, we consider as a performance index the ratio between the time needed by the "DWR" method or `CPLEX` to compute the "DWR" bound (respectively "auto" or "best") and the smaller between the two values. Normalization is then performed with respect to the fastest algorithm. For each value π on the horizontal axis, we report on the vertical axis the fraction of the dataset for which the corresponding algorithm is at most $\frac{1}{\pi}$ times slower than the fastest algorithm. The value on the vertical axis, corresponding to $\pi = 1$, indicates the fraction of the instances in which the considered method is the fastest among the two considered. As far as "DWR auto" is concerned, it outperforms `CPLEX` in around 8% of the instances, while "DWR best" in around 19%. From the figure, we can also see that in around 50% of the instances "DWR auto" is at most two orders of magnitude slower (see also discussion of Table 3 in Sect. 6.2), while "DWR best" only one order of magnitude.
In Fig. 15(b), we display the second performance profile based on the bounds of

the three methods. As a performance index we consider, for each instance and for each method, the ratio between the bound obtained by such a method and the best bound obtained using either "DWR auto," "DWR best," or CPLEX. Normalization is then performed with respect to the index of the best algorithm among the three considered. The picture reports the normalized performance index on the horizontal axis; for each value π on the horizontal axis, we report on the vertical axis the fraction of the dataset for which the corresponding algorithm closes at least a fraction π of the best achieved gap, on the considered set of 39 instances. We always assume that the LP relaxation value is available, thus the minimum closed gap for each instance and each algorithm is set to 0. For instance, if we look for a method which is able to close at least about 20% of the best achieved gap, then CPLEX is the best choice. However, if we increase this requirement to about 45% both "DWR auto" and "DWR best" outperform CPLEX. Finally, in almost 70% of the cases "DWR best" provides best bound, while both CPLEX and "DWR auto" give best bound in about 39% of the instances.



(a) Time **(b)** Gap

Fig. 15 Performance profiles comparing DWR auto, DWR best and CPLEX

7 Discussion

We have performed the first systematic investigation with an automatic Dantzig-Wolfe type reformulation of *arbitrary* MIPs. Even though it is clear from theory that such a reformulation can be used to improve the dual bound, it has not been considered a generally useful computational tool *in practice*. Thus, the most unexpected outcome of our study is that already a fairly basic implementation, combined with an automatic choice of the decomposition, is actually capable of competing with a state-of-the-art MIP solver. For some instances the dual bound computed by our method is so strong that, given a heuristic solution of optimal value, optimality can be proven at the root node. Furthermore, on a relevant subset of MIPLIB instances, we could automatically detect Dantzig-Wolfe type reformulations for which the decomposition approach yields an overall better computing behavior than a state-of-the-art branch-and-cut based general purpose solver.

The results even improve if we choose the decomposition by explicit computations, demonstrating that there is still potential. It turned out that different

decompositions for the same instance lead to, sometimes significantly, different dual bounds (see Figs. 13 and 14), and also to drastic differences in the computation times needed to solve the resulting relaxation. Thus, out of the many questions spawned by our work, the most important one is, both from a theoretical and a practical point of view, to characterize a *good decomposition*. We believe that answers will be hard to find as they immediately relate to the very core of computational integer programming: to better describe, in a computational and efficient manner, the convex hull of integer feasible points. On the other hand, approaching this important topic from a decomposition point of view may yield new insights previously overlooked.

Our experimental setup for detecting a matrix structure certainly can be improved; it is just *one* out of probably many conceivable approaches to come up with a proof-of-concept. In particular, the process of generating decompositions by simply changing parameters of partitioning algorithms, and then selecting one with best proxy measure, can give place to better and more sophisticated approaches. Many experimental issues on the design of good proxy measures are also left open, e.g., how the blocks' balancing and density impact on the overall performances.

Certainly, we will see alternatives in the future. We also alert the reader that the seeming omnipresence of arrowhead structure in MIPLIB instances (c.f. Fig. 16) may either reproduce structures which were incorporated in the model by a human modeler, accidentally or on purpose, or simply be an artifact of the model and algorithm we use to detect/enforce this structure. In any case, only a part of variables and constraints describe the logic and mechanism of the problem to be modeled. Another substantial part is present only because of "technical purposes" and "modeling tricks." Detecting and exploiting this information in a decomposition may lead to new insights into how a good MIP model should be formulated.

There are some possible immediate extensions concerning the implementation. Only further experimentation can show whether the advantage in the root node can be retained throughout the search tree (it is also conceivable that an advantage becomes visible *only* further down the tree). If one is only interested in a strong dual bound, the addition of generic cutting planes is a natural next step (see [16]).

Of course, at the moment, our work is not intended to produce a competitive tool, but to demonstrate that the direction is promising. Even in the future, we do not expect that decomposition techniques will become the single best option approach to solve MIPs. However, we hope that one can soon distinguish *a priori*, only based on the instance, whether it can pay to apply a decomposition like DWR or not. Our results indicate that promising instances are more than an exception.

Taking into account the fact that state-of-the-art solvers make successful use of generic cutting planes for about 15 years now, it is clear that outer approximations of the integer hull have a prominent headway in experience over inner approximations. We hope to have inspired further research and experimentation with the second option; indeed, follow-up research [14, 15, 24, 29] is already available.

A final word is in order on what to expect in terms of measuring the quality of a decomposition. The selection of "a good set" of cutting planes from a large pool of available ones is a core topic in the computational cutting plane literature. Theory suggests measures for the quality of *single* cutting planes, e.g., to prefer facet-defining inequalities or deep(er) cuts etc. For selecting from a collection of cuts, however, a theoretical argumentation is much scarcer and less-founded, and appears only problem-specific in the literature. In that light, a general theoretical *a*

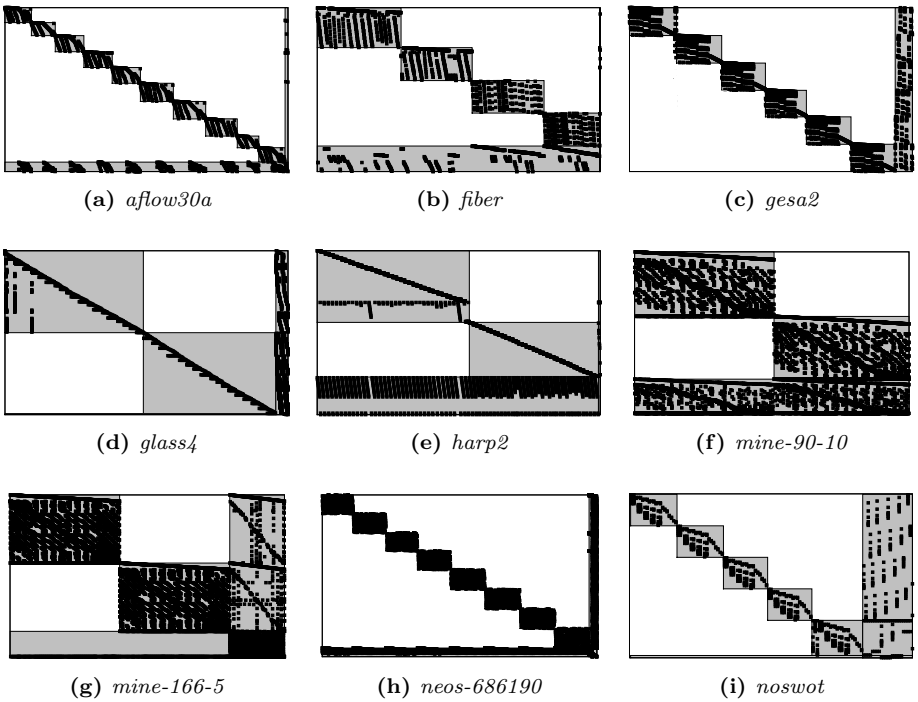
priori measure of what constitutes a good Dantzig-Wolfe decomposition currently appears to be out of reach, and we may need to fall back to computational proxies like the one we propose.

Acknowledgments. We sincerely thank an anonymous referee for thoughtful and motivating feedback, which led to a more meaningful experimental setup and a much improved organization of the material.

References

1. T. Achterberg, Th. Koch, and A. Martin. MIPLIB 2003. *Oper. Res. Lett.*, 34(4):361–372, 2006.
2. C. Aykanat, A. Pinar, and Ü.V. Çatalyürek. Permuting sparse rectangular matrices into block-diagonal form. *SIAM J. Sci. Comput.*, 25:1860–1879, 2004.
3. M. Bergner, A. Caprara, F. Furini, M.E. Lübbecke, E. Malaguti, and E. Traversi. Partial convexification of general MIPs by Dantzig-Wolfe reformulation. In O. Günlük and G.J. Woeginger, editors, *Integer Programming and Combinatorial Optimization*, volume 6655 of *Lect. Notes Comput. Sci.*, pages 39–51, Berlin, 2011. Springer.
4. R. Borndörfer, C.E. Ferreira, and A. Martin. Decomposing matrices into blocks. *SIAM J. Optim.*, 9(1):236–269, 1998.
5. A. Caprara, F. Furini, and E. Malaguti. Uncommon Dantzig-Wolfe reformulation for the temporal knapsack problem. *INFORMS Journal on Computing*, 25(3):560–571, 2013.
6. W.S. Cleveland. Robust locally weighted regression and smoothing scatterplots. *J. Amer. Statist. Assoc.*, 74(368):829–836, 1979.
7. M. Colombo, A. Grothey, J. Hogg, K. Woodsend, and J. Gondzio. A structure-conveying modelling language for mathematical and stochastic programming. *Math. Prog. Comp.*, 1:223–247, 2009.
8. G. Desaulniers, J. Desrosiers, and M.M. Solomon, editors. *Column Generation*. Springer-Verlag, Berlin, 2005.
9. J. Desrosiers and M.E. Lübbecke. A primer in column generation. In Desaulniers et al. [8], pages 1–32.
10. E.D. Dolan and J.J. Moré. Benchmarking optimization software with performance profiles. *Math. Programming*, 91:201213, 2002.
11. O. du Merle, D. Villeneuve, J. Desrosiers, and P. Hansen. Stabilized column generation. *Discrete Math.*, 194:229–237, 1999.
12. M.C. Ferris and J.D. Horn. Partitioning mathematical programs for parallel solution. *Math. Programming*, 80(1):35–61, 1998.
13. E. Fragnière, J. Gondzio, R. Sarkissian, and J.-Ph. Vial. A structure-exploiting tool in algebraic modeling languages. *Management Sci.*, 46:1145–1158, August 2000.
14. M.V. Galati and R. Pratt. The new decomposition algorithm in SAS/OR optimization. In *XXI International Symposium on Mathematical Programming*, 2012.
15. M.V. Galati, T.K. Ralphs, and J. Wang. Computational experience with generic decomposition using the DIP framework. In *Proceedings of RAMP 2012*. COR@L Laboratory, Lehigh University, 2012.
16. G. Gamrath and M.E. Lübbecke. Experiments with a generic Dantzig-Wolfe decomposition for integer programs. In P. Festa, editor, *Proceedings of the 9th Symposium on Experimental Algorithms (SEA)*, volume 6049 of *Lect. Notes Comput. Sci.*, pages 239–252, Berlin, 2010. Springer-Verlag.
17. A.M. Geoffrion. Lagrangean relaxation for integer programming. *Math. Programming Stud.*, 2:82–114, 1974.
18. G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel hypergraph partitioning: Application in VLSI domain. *IEEE Transactions on VLSI Systems*, 20(1), 1999.
19. G. Karypis and V. Kumar. hmetis 1.5: A hypergraph partitioning package. Technical report, Department of Computer Science, University of Minnesota,, 1998.
20. Th. Koch, T. Achterberg, E. Andersen, O. Bastert, T. Berthold, R.E. Bixby, E. Danna, G. Gamrath, A.M. Gleixner, S. Heinz, A. Lodi, H. Mittelmann, T. Ralphs, D. Salvagnin, D.E. Steffy, and K. Wolter. MIPLIB 2010 – Mixed Integer Programming Library version 5. *Math. Progr. Comp.*, 3(2):103–163, 2011.

21. M. Poggi de Aragão and E. Uchoa. Integer program reformulation for robust branch-and-cut-and-price algorithms. In *In Proceedings of the Conference Mathematical Program in Rio: A Conference in Honour of Nelson Maculan*, pages 56–61, 2003.
22. J. Puchinger, P.J. Stuckey, M.G. Wallace, and S. Brand. Dantzig-Wolfe decomposition and branch-and-price solving in G12. *Constraints*, 16(1):77–99, 2011.
23. T.K. Ralphs and M.V. Galati. DIP – decomposition for integer programming. `https://projects.coin-or.org/Dip`, 2009.
24. T.K. Ralphs, M.V. Galati, and J. Wang. DIP and DipPy: Towards a decomposition-based MILP solver. In *XXI International Symposium on Mathematical Programming*, 2012.
25. J. Tebboth and R. Daniel. A tightly integrated modelling and optimisation library: A new framework for rapid algorithm development. *Ann. Oper. Res.*, 104(1-4):313–333, 2001.
26. J.R. Tebboth. *A Computational Study of Dantzig-Wolfe Decomposition*. PhD thesis, University of Buckingham, 2001.
27. F. Vanderbeck. BaPCod – a generic branch-and-price code. `https://wiki.bordeaux.inria.fr/realopt/pmwiki.php/Project/BaPCod`, 2005.
28. F. Vanderbeck and L. Wolsey. Reformulation and decomposition of integer programs. In M. Jünger, Th.M. Liebling, D. Naddef, G.L. Nemhauser, W.R. Pulleyblank, G. Reinelt, G. Rinaldi, and L.A. Wolsey, editors, *50 Years of Integer Programming 1958–2008*. Springer, Berlin, 2010.
29. J. Wang and T.K. Ralphs. Computational experience with hypergraph-based methods for automatic decomposition in integer programming. In C. Gomes and M. Sellmann, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 7874 of *Lect. Notes Comput. Sci.*, pages 394–402, Berlin, 2013. Springer.
30. R.L. Weil and P.C. Kettler. Rearranging matrices to block-angular form for decomposition (and other) algorithms. *Management Sci.*, 18(1):98–108, 1971.



(a) *aflow30a* **(b)** *fiber* **(c)** *gesa2*

(d) *glass4* **(e)** *harp2* **(f)** *mine-90-10*

(g) *mine-166-5* **(h)** *neos-686190* **(i)** *noswot*

Fig. 16 Detected matrix structures for selected MIPLIB2003 and MIPLIB2010 instances

instance	opt*	LP	LP gap (%)	DWR auto gap			DWR best gap			CPLEX root			CPLEX default		
				gap (%)	closed (%)	time	gap (%)	closed (%)	time	gap (%)	closed (%)	time	nodes	time	ratio
10teams	924.00	917.00	0.76	0.05	92.86	2.22	**0.00**	100.00	1.46	**0.00**	100.00	0.47	0	0.21	10.57
aflow30a	1158.00	983.17	15.10	14.71	2.59	93.51	10.23	32.27	978.98	**5.35**	64.55	1.93	0	0.04	2337.75
aflow40b	1168.00	1005.66	13.90	13.90	0.00	70.52	13.90	0.00	550.81	**6.47**	53.44	4.15	0	0.04	1763.00
fiber	405935.00	156082.52	61.55	**1.07**	98.27	540.53	**1.07**	98.27	540.53	1.89	96.92	0.04	29	0.08	6756.63
fixnet6	3983.00	1200.88	69.85	18.89	72.96	2838.48	18.83	73.05	292.51	**6.06**	91.32	0.47	0	0.02	141924.00
gesa2	25779900.00	25476489.68	1.18	0.03	97.03	17.67	**0.03**	97.82	26.75	0.10	91.51	0.06	58	0.13	135.92
gesa2-o	25779900.00	25476489.68	1.18	0.78	33.53	113.14	**0.05**	96.16	183.64	0.21	82.38	0.08	0	0.03	3771.33
glass4	1200010000.00	800002400.00	33.33	33.33	0.00	3596.92	**8.33**	75.00	3.79	33.33	0.00	0.02	20646	10.37	346.86
harp2	-73899800.00	-74353341.50	0.61	0.61	0.00	1.59	**0.06**	0.00	0.33	0.37	39.60	0.05	0	0.03	53.00
manna81	-13164.00	-13297.00	1.01	0.08	92.05	535.49	0.04	96.23	4.41	**0.00**	100.00	0.11	0	0.11	4868.09
mkc	-563.85	-611.85	8.51	**0.16**	98.13	106.25	**0.16**	98.13	106.25	3.78	55.63	0.32	23110	43.86	2.42
modglob	20740500.00	20430947.62	1.49	0.79	47.30	3603.56	**0.12**	91.74	3501.33	0.14	90.48	0.03	0	0.01	360356.00
noswot	-41.00	-43.00	4.88	**0.49**	90.00	1.26	**0.49**	90.00	1.26	4.88	0.00	0.01	6902434	731.62	0.00
opt1217	-16.00	-20.02	25.13	25.13	0.00	0.07	25.13	0.00	0.02	0.00	100.00	0.07	0	0.01	7.00
p2756	3124.00	2688.75	13.93	**0.27**	98.05	234.98	**0.27**	98.07	111.69	7.47	46.41	0.07	199	0.20	1174.90
pp08a	7350.00	2748.35	62.61	2.50	96.01	0.47	**2.27**	96.38	5.74	2.52	95.97	0.08	4	0.09	5.22
pp08aCUTS	7350.00	5480.61	25.43	2.58	89.86	0.58	**2.27**	91.09	9.72	3.82	84.97	0.08	224	0.23	2.52
rout	1077.56	981.86	8.88	**0.68**	92.34	5.90	**0.68**	92.34	5.90	8.86	0.26	0.06	8921	8.74	0.68
set1ch	54537.80	32007.73	41.31	0.10	99.76	0.61	**0.07**	99.84	0.88	0.92	97.77	0.07	511	0.34	1.79
timtab1	764772.00	28694.00	96.25	16.40	82.96	244.59	**14.60**	84.84	326.55	39.05	59.43	0.15	38451	50.04	4.89
timtab2	1096560.00	83592.00	92.38	36.50	60.49	3578.98	**25.26**	72.66	2178.64	46.00	50.20	0.31	32917	85.96	41.64
tr12-30	130596.00	14210.43	89.12	89.12	0.00	3586.64	2.83	96.83	839.54	**0.68**	99.24	0.59	0	0.02	179332.00
vpm2	13.75	9.89	28.08	**1.71**	93.92	1.05	**1.71**	93.92	1.05	6.44	77.05	0.00	1632	0.46	2.28
Average			30.28	11.29	62.53		5.58	77.16		7.75	68.57				
< 1															8.70
< 10															39.13
$< 10^2$															52.17
beasleyC3	754.00	40.43	94.64	15.38	83.74	3600.56	15.38	83.74	3600.00	**8.24**	91.29	1.06	0	0.38	9475.16
csched010	408.00	332.42	18.52	**7.91**	57.28	1041.23	**7.91**	57.28	1041.23	12.62	31.86	0.55	25753	165.15	6.30
enlight13	71.00	0.00	100.00	**57.55**	42.45	1068.72	**57.55**	42.45	1068.72	80.66	19.34	1.37	17828	149.47	7.15
gmu-35-40	-2406677.75	-2406943.56	0.01	0.01	0.00	0.12	**0.01**	0.00	0.09	**0.01**	2.54	0.04	0	0.03	4.00
m100n500k4r1	-24.00	-25.00	4.17	**4.17**	0.00	87.10	**4.17**	0.00	0.02	**4.17**	0.00	0.06	0	0.02	4355.00
macrophage	374.00	0.00	100.00	0.37	99.63	3599.35	**0.05**	99.95	1385.91	11.48	88.52	17.31	8522	142.72	25.22
mcsched	211913.00	193774.75	8.56	**0.06**	99.29	84.30	**0.06**	99.29	84.30	8.54	0.23	1.03	110052	367.61	0.23
mine-166-5	-566395707.87	-821763677.67	45.09	**7.33**	83.74	1426.12	**7.33**	83.74	1426.12	22.79	49.44	1.81	149	2.18	654.18
mine-90-10	-784302337.63	-887165318.51	13.12	**0.49**	96.24	2983.32	**0.49**	96.24	2983.32	8.73	33.41	0.85	22078	59.55	50.10
neos-686190	6730.00	5134.81	23.70	23.70	0.00	3594.74	**17.60**	25.73	4.84	21.54	9.11	0.57	0	0.08	44934.25
pigeon-10	-9000.00	-10000.00	11.11	**11.11**	0.00	44.13	**11.11**	0.00	1.32	**11.11**	0.00	0.01	0	0.02	2206.50
pw-myciel4	10.00	0.00	100.00	60.00	40.00	27.50	**0.00**	100.00	3600.00	53.54	46.46	1.87	0	0.09	305.56
ran16x16	3823.00	3116.43	18.48	**10.77**	41.71	0.93	**10.77**	41.71	0.93	6.93	62.51	0.61	0	0.02	46.50
reblock67	-34627142.08	-39339910.92	13.61	**0.13**	99.02	529.73	**0.13**	99.02	529.73	8.87	34.86	0.63	489311	488.15	1.09
rmine6	-457.19	-462.31	1.12	**0.42**	62.23	836.89	**0.42**	62.23	836.89	0.98	12.26	0.60	1547	9.70	86.28
rococoC10-001000	11460.00	7515.27	34.42	34.43	0.00	13.75	34.42	0.00	3.87	**11.62**	66.25	0.58	0	0.07	196.43
Average			36.66	14.61	50.33		10.46	55.71		16.99	34.26				
< 1															6.25
< 10															31.25
$< 10^2$															56.25

Table 3 Comparison of the root node dual bounds provided by DWR and CPLEX: in particular we report the percentage of instances for which DWR was faster than CPLEX (< 1), within one order of magnitude (< 10), and within two orders of magnitude ($< 10^2$).