Partial Convexification of General MIPs by Dantzig-Wolfe Reformulation

Martin Bergner[o], Alberto Caprara[*], Fabio Furini[*],
Marco E. Lübbecke[o], Enrico Malaguti[*], Emiliano Traversi[*]

[*]: DEIS, Università di Bologna,
Viale Risorgimento 2, I-40136 Bologna, Italy.
E-mail:{alberto.caprara,fabio.furini,enrico.malaguti,emiliano.traversi2}@unibo.it

[o]: Chair of Operations Research, RWTH Aachen University,
Templergraben 64, D-52056 Aachen, Germany.
E-mail:{martin.bergner,marco.luebbecke}@rwth-aachen.de

Abstract

Dantzig-Wolfe decomposition is well-known to provide strong dual bounds for specially structured mixed integer programs (MIPs) in practice. However, the method is not part of any state-of-the-art MIP solver: it needs tailoring to the particular problem; the typical bordered block-diagonal matrix structure determines the decomposition; the resulting column generation subproblems need to be solved efficiently; branching and cutting decisions need special attention; etc. We perform an extensive computational study on the 0-1 dynamic knapsack problem (without block-diagonal structure) and on general MIPLIB2003 instances in order to (in-)validate such reservations against the method. We present a tool which, given an LP file, automatically detects an exploitable matrix structure, accordingly performs a Dantzig-Wolfe type reformulation of subsets of the constraints (partial convexification), and performs column generation to obtain an optimal LP relaxation. Our results strongly support that Dantzig-Wolfe decomposition holds more promise as a general-purpose tool than previously acknowledged by the research community.

1 Introduction

A considerable, if not the major, part of the computational (mixed) integer programming machinery is about outer approximating the convex hull of integer feasible points (or mixed integer sets). The addition of valid inequalities, a.k.a. cutting planes, is the traditional general-purpose device which proved powerful in strengthening the linear programming relaxations. Now, given that the integer hull is the ultimate object of desire, we ask: Why don't we just work with it? Being fully aware of the boldness of this question, we want to seriously re-consider it in this work by *explicitly* constructing parts of the integer hull via a generic Dantzig-Wolfe type reformulation. This extends previous *partial convexification* approaches which only separate a subset of facets from the integer hull.

Dantzig-Wolfe reformulation (or decomposition) of mixed integer programs (MIPs) became a computationally very successful—sometimes the only applicable—approach to producing high-quality solutions for well-structured combinatorial optimization problems like vehicle routing, cutting stock, p-median, generalized assignment, and many others. Their common structure is the (bordered) block-diagonal form of the coefficient matrix, the traditional realm of Dantzig-Wolfe decomposition. Be aware that so far its use is tailored to the application and 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 *how* it looks like and *how* to exploit it algorithmically. In particular in view of the automatism with which general-purpose cutting planes are separated in all serious MIP solvers, this is an unsatisfactory situation. This immediately raises several research questions of increasing ambition:

- When the MIP contains a *known structure* suitable to Dantzig-Wolfe reformulation (DWR), can *an algorithm* detect (and of course: exploit) it?
- When the contained structure is *unknown*, can it still be detected (and of course: exploited)?
- When it is known that the MIP *does not contain* a structure particularly suitable to DWR in the traditional meaning, can DWR still be a useful computational tool?

Besides our work, we are not aware of any attempts to systematically answer the first two questions, and it can be taken as a fact that the research community currently is very inclined to answer the last, and most interesting question in the negative. In our extensive computational study on several of the hardest MIPLIB2003 instances, we do not only suggest first attempts to accomplish the structure detection. We also support the DWR's potential of becoming a general-purpose method for improving dual bounds by giving surprisingly encouraging computational results. Of course, at the moment, our work is not intended to produce a competitive tool, but to provide a proof-of-concept and demonstrate that the direction is promising. The main findings of our work can be summarized as follows:

- A known or hidden double-bordered block-diagonal (so-called: *arrowhead*) matrix structure can be effectively recovered by a suitable use of (hyper-)graph partitioning algorithms.
- For the dynamic 0-1 knapsack problem, the natural formulation of which does not expose any bordered block-diagonal structure, we give impressive computational results which demonstrate the potential of our method for closing the integrality gap.
- For some of the hardest MIPLIB2003 instances our reformulations produce stronger dual bounds than `CPLEX` 12.2 with default cutting planes enabled, in comparable or less CPU time.
- As the number of possible reformulations is vast, some guidance is much needed. We evaluate *a posteriori* measures and analyze their impact on the quality of decompositions. This way, we not only conclude that a reformulation may imply a computational advantage, but we also obtain some hints on when *it does not*.

We stress again that we provide a general-purpose implementation which reads any LP file and automatically performs the detection, the reformulation, and the column generation itself in order to obtain a strong LP relaxation, without any user interaction.

The flow of the paper is as follows: We briefly introduce the concept of partial convexification, and the overall approach. This is then applied first to a problem not containing a matrix structure directly amenable to classical Dantzig-Wolfe reformulation, and then to general MIPs. We report on extensive computational experiments and close with a discussion of our results.

1.1 Partial Convexification and Dantzig-Wolfe Reformulations

Consider a MIP of the form

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

Let $P := \{x \in \mathbb{Q}^n : Dx \leq e\}$ and $P_{IP} := \text{conv}\{P \cap \mathbb{Z}^{n-q} \times \mathbb{Q}^q\}$ denote the LP relaxation and the integer hull with respect to constraints $Dx \leq e$, respectively. We assume for ease of presentation that P is bounded. The *Dantzig-Wolfe reformulation* (DWR) of constraints $Dx \leq e$ (which in fact is

nothing else but their reformulation according to the theorems by Minkowski and Weyl on the representations of convex polyhedral cones) is obtained by expressing $x \in P_{IP}$ as a convex combination of the vertices V of P_{IP}, namely

$$\max\{c^t x : Ax \leq b,\ x = \textstyle\sum_{v \in V} \lambda_v v,\ \sum_{v \in V} \lambda_v = 1,\ \lambda_v \geq 0\ (v \in V),\ x \in \mathbb{Z}^{n-q} \times \mathbb{Q}^q\}. \qquad (2)$$

It is well-known that the resulting LP relaxation is stronger than that of (1) when $P_{IP} \subsetneq P$, which is the main motivation of performing the reformulation. This *partial convexification* with respect to the constraints $Dx \leq e$ corresponds to adding (implicitly) *all* valid inequalities for P_{IP} to (1), which in a sense is the best one can hope for.

The reformulated MIP (2) has fewer constraints remaining, the so-called *master constraints* $Ax \leq b$, plus the convexity constraint and the constraints linking the *original* x variables to the *extended* λ variables. On the downside of it, in general MIP (2) has an exponential number of λ variables, so its LP relaxation is solved by column generation, where the pricing or *slave MIP* problem to check whether there are variables with positive reduced cost to be added to the current *master LP* problem calls for the optimization of a linear objective function over P_{IP}. This slave MIP can either be solved by a general-purpose solver or by a tailored algorithm to exploit a specific structure, if known.

In the classical DWR setting, k disjoint sets of constraints are partially convexified, namely when the matrix D has block-diagonal form

$$D = \begin{bmatrix} D^1 & & & \\ & D^2 & & \\ & & \ddots & \\ & & & D^k \end{bmatrix},$$

where $D^i \in \mathbb{Q}^{m_i \times n_i}$ for $i = 1, \ldots, k$. In other words, $Dx \leq e$ decomposes 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 partially convexified in the above spirit. The constraints in different blocks need not even be disjoint. We call k the *number of blocks* of the reformulation. 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 looks like this

$$\begin{bmatrix} D^1 & & & & F^1 \\ & D^2 & & & F^2 \\ & & \ddots & & \vdots \\ & & & D^k & F^k \\ G^1 & G^2 & \cdots & G^k & H \end{bmatrix}.$$

The constraints associated with the of rows of G^1 are called the *coupling constraints* and the variables associated with the columns of F^1 are called the *linking variables*. One can get to a form without linking variables (and with additional linking constraints) by replacing each linking variable by one copy for each nonzero entry of the associated column and adding constraints imposing that all these copies be equal. Then we are back to the above traditional setting.

1.2 Related Literature

For a general background on Dantzig-Wolfe decomposition of MIPs we refer to the recent survey [14]. The notion of *partial convexification* has been used before. For instance, Sherali et al. [12] choose

small subsets of integer variables (and usually only one constraint) to directly separate cutting planes from the partial convexification P_{IP}. Our approach goes far beyond that in terms of number of constraints and variables involved in the reformulation.

There are several implementations which perform a Dantzig-Wolfe decomposition of a general MIP, and handle the resulting column generation subproblems in a generic way, like `BaPCod` [13], `DIP` [11], `G12` [10], and `GCG` [8]. In [8] it is also shown that a generic reformulation algorithm performs well when a block-diagonal matrix structure is *known and given* to the algorithm. However, to the best of our knowledge, there is no code which automatically detects a possible decomposition structure (or creates one by variable splitting), let alone exploits it.

Bordered block-diagonal matrices play an important role in, e.g., numerical linear algebra. One typically tries to identify a fixed number of blocks of almost equal size with as few constraints in the border as possible. The motivation is to prepare a matrix for parallel computation, like for solving linear equation systems, see, e.g., [2], and the references therein.

We would also like to note the following possible connection to multi-row cuts which recently received some attention. Computational experiments [6] suggest that obtaining valid inequalities from more than one row of the simplex tableau holds some potential. However, in order to use this potential it seems to be imperative to have a criterion for which rows to select. As our choice of which rows to convexify is equally important for similar reasons, the lessons we learn may help there as well.

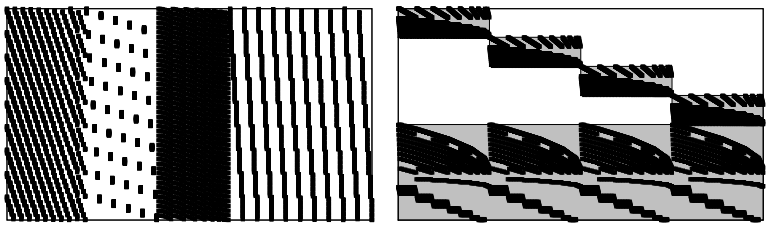
2 Automatic Detection of an Arrowhead Form

As mentioned, permuting a matrix into arrowhead form is a common topic in numerical linear algebra. There is a folklore connection between a matrix and an associated (hyper-)graph which is also used in the graph partitioning approach by Ferris and Horn [7]. We first briefly illustrate their algorithm and then adapt it to our purpose of finding tighter LP relaxations of MIPs through DWR.

Assume the number k of blocks is given. Consider the bipartite graph G with one node r_i associated with each row i of A, one node c_j associated with each column j of A, and one edge (r_i, c_j) whenever $a_{ij} \neq 0$. Assuming the number $m + n$ of nodes is a multiple of k, find a *min k-equicut* of G, i.e. partition its node set into k subsets of equal size so that the number of edges that are cut, i.e. that join nodes in different subsets, is minimized. Finally, remove a set of nodes of G so as to ensure, for the remaining graph, no edge is joining nodes in different subsets of the partition. This yields the final arrowhead form: the row nodes removed represent the coupling constraints, the column nodes removed represent the linking variables, and the remaining row and column nodes in the ith subset of the partition define the submatrix D^i. In practice, min k-equicut is solved heuristically by using `Metis` which is an implementation of the multilevel graph partitioning algorithm in [9]. The final removal is done greedily by iteratively removing the node joined to nodes in other subsets by the largest number of edges. Moreover, in order not to require that the number of nodes be a multiple of k nor to insist in having subsets of the same size, dummy nodes disconnected from the rest of the graph are added before doing the partitioning. This imposes only an upper bound on the size of each subset and on the number of subsets (noting that subsets with dummy nodes only are irrelevant in the end).

We use a variant of this method as we would like to avoid the final removal phase, having an explicit cost in the graph partitioning phase in case a given constraint becomes a coupling constraint or a given variable becomes a linking variable. Specifically, we define the hypergraph H with a *node* for each nonzero entry of A. There is a *constraint hyperedge* joining all nodes for nonzero entries in the ith

row of A. Moreover, there is a *variable hyperedge* joining all nodes for nonzero entries in the jth column of A. To each hyperedge we associate a cost as discussed above. Then, we find a heuristic min k-equicut of H, now with the objective of minimizing the sum of the costs of the hyperedges that are cut, i.e. that join nodes in different subsets of the partition. Now the solution already defines the arrowhead form, as the constraint hyperedges that are cut represent the coupling constraints, the variable hyperedges that are cut represent the linking variables, and the remaining constraint and variable hyperedges joining only nodes in the ith subset of the partition define the submatrix D^i. Also in this case we use dummy nodes for allowing for varying block sizes.



(a) original *10teams* (b) detected arrowhead

Figure 1: Matrix structure directly from the LP file (*10teams*) and with arrowhead structure detected

3 Computational Results

3.1 Notes on the Implementation and Experimental Setup

All experiments were done on an Intel i7 quad-core PC with 8GB main memory running Linux 2.6.34 (single thread). As MIP and LP solver we used `CPLEX` 12.2 (single thread). We determined the constraints to reformulate by the method of Section 2, solving the min k-equicut by `Hmetis` [9].

Concerning the implementation, the overall algorithm detects an arrowhead form in the matrix (which is always present), splits linking variables in order to obtain a bordered block-diagonal form, and then performs a Dantzig-Wolfe reformulation of the resulting blocks. The only special feature of the column generation is the use of a dual variable stabilization [5] where the box around the stability center is narrowed to a fixed small percentage of the absolute duality gap, whenever the dual bound improves (and the stability center is re-located). This consistently improved running times.

3.2 A Class of Structured MIPs without Block-Diagonal Form

We first consider a class of problems which has a natural MIP formulation with a coefficient matrix which is very structured, but not bordered block-diagonal at all. A *dynamic MIP* extends a MIP

$$\max\{c^t x : Ax \leq b,\ x \in \mathbb{Z}^{n-q} \times \mathbb{Q}^q\}$$

by a time horizon $[0, T]$ and a time interval $[s_j, f_j] \subseteq [0, T]$ in which each variable x_j is *active*. In the dynamic MIP all constraints must be satisfied at any instant in the time horizon restricting attention to the variables that are active at that instant (with the other variables fixed to 0). It is elementary to observe that the check can be restricted to the instants $I := \{s_1, \ldots, s_n, f_1, \ldots, f_n\}$ in which a

variable becomes active or inactive, yielding the following MIP formulation of the dynamic MIP:

$$\max\{c^t x : A^i x \leq b \ (i \in I),\ x \in \mathbb{Z}^{n-q} \times \mathbb{Q}^q\}, \tag{3}$$

where A^i is obtained from A by setting to 0 all the entries of columns associated with variables not active at $i \in I$.

A simple example of a dynamic MIP is the the *dynamic 0-1 knapsack* problem, in which $q = 0$ and $Ax \leq b$ takes the form $a^t x \leq b,\ 0 \leq x \leq 1$ for some $a \in \mathbb{Q}^n_+$ and $b \in \mathbb{Q}_+$. In words, we have a set of items each with a size a_j and active for time interval $[s_j, f_j]$ and a container of capacity b and we must select a set of items to pack in the container (for the whole interval in which they are active) so that the total size packed in each instant is at most b. This problem is also known as *resource allocation* or *unsplittable flow on a line*. The latter name is due to the fact that the time dimension may in fact be a (one-dimensional) spatial dimension, as it is the case in the following real-world application that we encountered in railway service design [4]. We are given a railway corridor with m stations $1, \ldots, m$ and n railcars, the jth having weight a_j and to be transported from station s_j to station f_j, gaining profit c_j in this case. Moreover, we have a train that travels from station 1 to station m and can carry at the same time railcars for a total weight b. The problem is to decide which railcars the train carries in order to maximize the profit, and is clearly a dynamic 0-1 knapsack problem.

Although the dynamic 0-1 knapsack problem has been widely studied in the literature, the most basic questions on its complexity are still open, namely it is not known whether it is strongly NP-hard (being trivially weakly NP-hard as it generalizes the 0-1 knapsack problem) or it has a polynomial-time approximation scheme.

In Table 1 we report the results obtained on a hard representative instance in each of the 0-1 dynamic knapsack classes considered in [3] when the parameter k varies. This parameter is chosen in such a way to enforce a decomposition in blocks of equal size of 128, 64, and 32 constraints each (this proved to provide best bounds as compared to computation times; for a larger number of much smaller blocks, computation times went up considerably).

The next question is how well DWR behaves with respect to the application of a general-purpose MIP solver to the original formulation. To this end, we developed a very basic depth-first branch-and-price method branching on the most fractional *original* variable and compared it to `CPLEX` 12.2 in Table 2. We report the time required by branch-and-bound and, in case this exceeds the time limit of one hour, the gap between the best bound and the best solution found. The table shows the clear advantage of using the DWR with the choice of parameter k, as all instances can be solved to optimality within minutes, whereas the solver reaches the one-hour time limit for all instances except I65 (where profits equal volume). The results on the other instances in the six classes are analogous.

3.3 MIPLIB2003

In order to assess the generality of the proposed method we tested the algorithm on MIPLIB2003 instances [1]. One third of all instances are either too difficult (the LP relaxation could not be computed with any decomposition we tested) or too big (e.g., linking variables occurred so often that variable splitting resulted in too many master constraints). For the remaining we are always capable of improving over the LP relaxation in terms of dual bound. Moreover, for eight instances we considerably improve over the root node with default cuts, see Table 3.

The following different sets of penalties on hyperedges for splitting continuous and discrete variables, coupling constraints, and the maximal number of blocks were used. In order to allow for blocks of

inst	k	ℓ	c	max var	max cons	cont	time (cont)	opt	+time (B&P)
I45	10	243	0	421	149	50325.00	94.36	50324	147.53
I55	10	189	0	1011	155	120505.00	252.98	120505	0.01
I65	10	243	0	448	157	37020.00	8.95	37016	2.98
I75	10	199	0	608	159	66196.50	38.91	66179	323.32
I85	10	159	0	1004	146	119234.00	121.89	119233	59.09
I95	10	243	0	621	144	68432.00	76.86	68432	0.00
I45	20	505	0	241	79	50327.50	33.75	50324	77.31
I55	20	402	0	594	88	120539.75	179.73	120505	2303.58
I65	20	509	0	241	79	37019.50	7.31	37016	8.97
I75	20	430	0	353	84	66185.00	30.02	66179	67.82
I85	20	344	0	576	83	119234.00	55.62	119233	49.97
I95	20	515	0	362	82	68432.00	29.21	68432	0.01
I45	30	977	0	135	39	50347.50	25.5	50324	153.41
I55	30	825	0	321	46	120529.00	65.75	120505	1126.21
I65	30	976	0	139	41	37016.00	10.37	37016	0.00
I75	30	880	1	192	43	66199.50	17.63	66179	150.19
I85	30	756	0	307	42	119263.00	35.71	119233	514.87
I95	30	1041	0	200	42	68434.50	20.87	68432	9.19

Table 1: Performance of DWR on the dynamic 0-1 knapsack problem. Listed are the number k of blocks, the number ℓ of linking variables and number c of coupling constraints, the maximum numbers of variables and constraints per block, the value of the continuous relaxation and the time to obtain it, the optimum, and the additional solution time which our simple-minded branch-and-price algorithm needs (in addition to the root node) to solve the instance to optimality.

inst	cols	rows	opt	cont	time	root/cuts	time	lb*	ub*	time
I45	3433	1280	50324	58235.01	0.02	50981.01	2.72	50324	50499.09	TL
I55	8266	1280	120505	137308.63	0.03	121274.42	4.53	120499	120694.37	TL
I65	3434	1280	37016	41155.00	0.02	37052.12	1.5	37016	37016.00	1.67
I75	4771	1280	66179	75300.45	0.02	66705.48	2.63	66178	66197.50	TL
I85	8656	1280	119233	134801.82	0.03	119721.88	4.53	119233	119233.00	1357.88
I95	5209	1280	68432	76853.31	0.03	68949.04	3.01	68432	68469.51	TL

Table 2: Performance of the general-purpose MIP solver `CPLEX` for the dynamic 0-1 knapsack problem. Listed is the instance name, the number of variables and constraints, the optimum, the value of the continuous relaxation and the time to obtain it, the objective value with default cuts added and the time to obtain it, and the primal and dual bounds at the time limit (TL) of one hour or at optimality.

unequal size, the portion of dummy variables ranged over 0.0, 0.2, and 0.5.

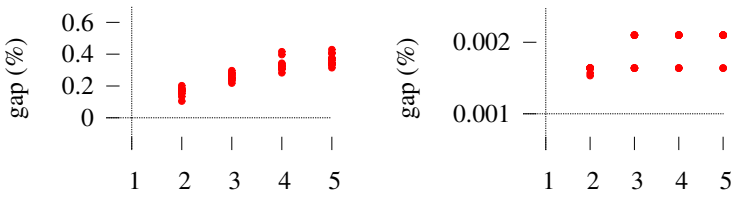
contin. var	discr. var	cons	blocks k
1	2	5	$2, \ldots, 5, 30$
1	1	1	$2, \ldots, 5, 30$
1	2	10^6	$2, \ldots, 5, 30$

				DWR reformulation					CPLEX 12.2					
inst	cols	rows	opt	k	ℓ	c	bound	time	cont	time	root/cuts	time	best	time
10teams	2025	230	924	4	450	110	**924**	**2.74**	917	0.03	**924**	0.98	924	3.69
mkc	5325	3411	563.8	4	38	40	**565.0**	**21.40**	611.8	0.02	585.1	0.44	564.7	TL
noswot	128	182	41.0	2	0	1	**41.2**	**5.19**	43.0	0.00	43.0	0.02	41.0	641.51
p2756	2756	755	3124	3	0	16	**3115.6**	20.70	2688	0.01	2947	0.11	3124	**0.90**
pp08a	240	136	7350	2	0	0	**7190**	6.00	2748	0.00	7164	0.10	7350	**0.54**
timtab1	397	171	764772	2	0	0	**642803**	**124.85**	28694	0.00	466129	0.19	764772	TL
timtab2	675	294	1096560	3	23	0	**830606**	**1913.82**	83592	0.00	592137	0.45	797765	TL
vpm2	378	234	13.7	2	14	12	**13.4**	1.02	9.8	0.00	12.8	0.01	13.7	**0.4**

Table 3: Comparison of the bounds provided by our automatic DWR reformulation approach and and a general-purpose MIP solver for eight selected instances of MIPLIB2003. The headings have the same meanings as in the previous tables.

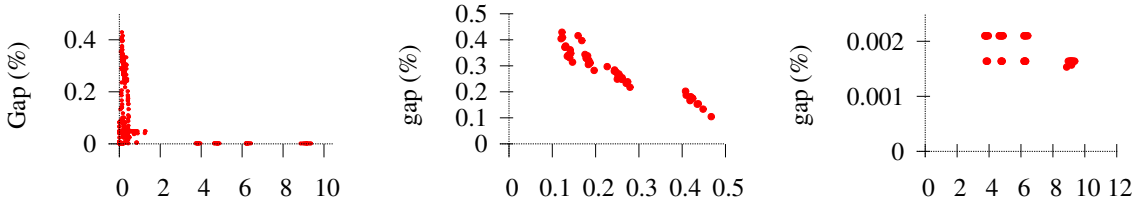
Table 3 shows that in all cases the bound found by our DWR approach is much better than the continuous and root node bounds. In cases *p2756* and, to a lesser extent, *pp08a* the time required to compute our bound is not competitive with the time required by the general-purpose solver to solve the instance. These two times are comparable in cases *10teams* and *vpm2*. On the other hand, for cases *mkc*, *noswot*, *timtab1*, and *timtab2* the time to compute our (partly much better) bound is significantly shorter than the solver branch-and-bound time. This stimulates further investigation of our approach for these instances. Note that we explicitly keep the linking constraints between original and extended variables, producing a significant overhead, and still we are able to compete.

A posteriori measures for the quality of a decomposition. It sounds so easy: Choose a decomposition and perform a partial convexification of the blocks. In practice however, when there is no apparent matrix structure, it is by no means clear how a decomposition should be chosen. We therefore evaluated the impact of a few natural parameters of the decomposition on the percentage of the integrality gap closed. In summary, there are only two parameters which show a significant influence; these are (1) the number of blocks (Fig. 2), and (2) the (normalized) average integrality gap of the blocks (Fig. 3). The more blocks are reformulated the potentially weaker the resulting dual bound. On the other hand, computation times drastically increase when the number of blocks is very/too small. Precise numbers of course depend on the instance. For the average integrality gap in the blocks the trend is the smaller the gap, the larger the percentage of gap closed by the DWR, see Fig. 3(a). This is to be expected, but has to be taken with care, see Figs. 3(b) and (c): For the *timetab1* instance, with large absolute objective function values and thus a large spectrum of possible values, this trend shows almost ideally; while *mkc* has relatively small absolute objective function values, and there are essentially two decomposition qualities, which are not separated. Results are not consistent for other parameters like number of coupling constraints or linking variables, number of integer variables in each block, or angle of the objective function's gradient with the gradients of constraints.



(a) *timtab1* (b) *mkc*

Figure 2: Number of blocks against integrality gap of the dual bound obtained by DWR



(a) all (b) *timtab1* (c) *mkc*

Figure 3: Average integrality gap of the slaves (normalized by the respective optimum) versus integrality gap of the dual bound obtained by DWR. *All* instances are those selected in Table 3.

From this experience we can give the following computational advise: When looking for a *good* number of blocks, start with many. As long as this is computationally easy, decrease the number of blocks. We note that the two relevant quality measures can be evaluated even *a priori* before the actual decomposition is performed.

4 Discussion

We have performed the first systematic computational study with an automatic partial convexification by a Dantzig-Wolfe type reformulation of subsets of rows of *arbitrary* mixed integer programs. While it is clear from theory that a partial convexification can 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 a careful choice of the decomposition, is actually capable of competing with or even beating a state-of-the-art MIP solver. Interestingly, to the best of our knowledge for the first time, the "careful choice of the decomposition" is done by an algorithm, and is not given by the user.

One should not conceal the fact that an inner approximation of the integer hull still has considerable disadvantages. In contrast to cut generation, which can be stopped whenever it no longer proves effective, we must perform column generation essentially until the end in order to retain a valid dual bound. Also, the process is not reversible as we cannot easily get rid of the extended formulation. Lastly, the choice of the decomposition is final. This "single shot" contrasts cutting plane algorithms which can iteratively increase the number of classes of valid inequalities considered.

There are some possible immediate extensions concerning the implementation. Even though we

have mainly solved the LP relaxation so far, our tool is already able to perform a true branch-and-price. Only further experimentation can show whether the advantage in the root node can be retained throughout the search tree, not only for dynamic MIPs but also for general MIPs (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.

All the questions initially posed in the introduction are computationally and conceptually extremely hard, and at present one cannot hope for conclusive answers to any of them. We therefore think that our work spawns a number of interesting research directions worthwhile to further pursue.

1. The most important task, both from a theoretical and a practical point of view, is to characterize a *good decomposition*. This can also help in quickly deciding whether it is worth trying a reformulation or not.
2. We have seen that the matrix needs not contain any (known) apparent structure in order to make the method perform well. In particular our third question from the introduction needs reformulation in the light of our results: what does it mean that a model be suitable for application of DWR?
3. *Extended formulations* are a topic on its own in combinatorial optimization, mainly used as a theoretical vehicle to obtain stronger formulations. As DWR is a particular kind of extended formulation it is natural to ask: Can an approach like ours turn this into a computational tool?
4. We complained that a once chosen decomposition is static. Is there a computationally viable way for dynamically updating an extended formulation, like our DWR?

Taking into account that state-of-the-art solvers make successful use of cutting planes since more than 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 on the topic of this paper.

References

[1] T. Achterberg, T. 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] A. Caprara, F. Furini, and E. Malaguti. Exact algorithms for the temporal knapsack problem. Technical report OR-10-7, DEIS, University of Bologna, 2010.

[4] A. Caprara, E. Malaguti, and P. Toth. A freight service design problem for a railway corridor. *Tran. Sci.*, 2010.

[5] O. du Merle, D. Villeneuve, J. Desrosiers, and P. Hansen. Stabilized column generation. *Discr. Math.*, 194:229–237, 1999.

[6] D.G. Espinoza. Computing with multi-row Gomory cuts. *Oper. Res. Lett.*, 38:115–120, 2010.

[7] M.C. Ferris and J.D. Horn. Partitioning mathematical programs for parallel solution. *Math. Program.*, 80(1):35–61, 1998.

[8] 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.

[9] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Comput.*, 20(1):359–392, 1998.

[10] J. Puchinger, P.J. Stuckey, M.G. Wallace, and S. Brand. Dantzig-Wolfe decomposition and branch-and-price solving in G12. *Constraints*, 2010. To appear.

[11] T.K. Ralphs and M.V. Galati. DIP – decomposition for integer programming. `https://projects.coin-or.org/Dip`, 2009.

[12] H.D. Sherali, Y. Lee, and Y. Kim. Partial convexification cuts for 0-1 mixed-integer programs. *European J. Oper. Res.*, 165(3):625–648, 2005.

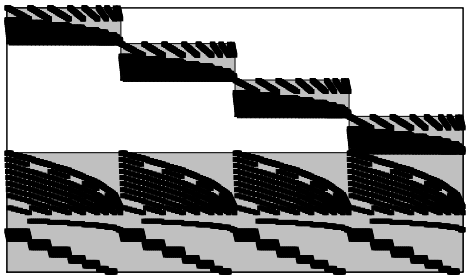
[13] F. Vanderbeck. BaPCod – a generic branch-and-price code. `https://wiki.bordeaux.inria.fr/realopt/pmwiki.php/Project/BaPCod`, 2005.

[14] 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.

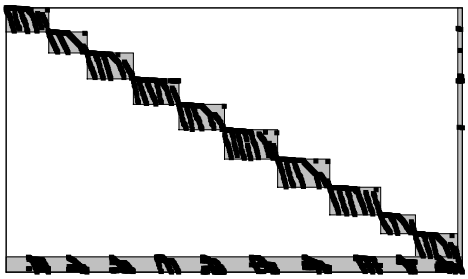
,

inst	rows	cols	k	ℓ	c	max vars	max cons	cont	time	opt	cont	time	root/cuts	time
10teams	230	2025	4	0	110	450	30	924.00	2.74	924	917.00	0.03	924.00	0.98
aflow30a	479	842	10	9	28	98	53	987.77	10.7	1158	983.17	0	1096.05	2.13
arki001	1048	1388	20	89	184	106	54	7580109.44	1304.39	7580810	7579599.81	0.06	7580007.82	0.88
fiber	363	1298	4	6	69	389	95	308023.57	149.58	405935	156082.52	0.01	398760.67	0.07
fixnet6	478	878	10	6	15	102	54	3220.58	19.99	3983	1200.88	0	3741.79	0.51
gesa2	1392	1224	6	80	0	228	237	25779856.37	78.46	25779900	25476489.68	0.01	25754151.33	0.1
gesa2-o	1248	1224	30	125	23	66	53	25698200.33	740.45	25779900	25476489.68	0.01	25726580.48	0.12
glass4	396	322	2	14	3	170	198	909353202.49	777.61	1200010000	800002400.00	0	800002400.00	0.03
harp2	112	2993	2	481	26	1957	49	-74144778.48	24.83	-73899800	-74353341.50	0.01	-74173715.03	0.08
liu	2178	1156	100	0	13	47	24	365.56	1255.63	*	346.00	0.01	560.00	0.67
misc07	212	260	1	31	37	260	196	2372.50	36.35	2810	1415.00	0	1425.00	0.06
mkc	3411	5325	2	5	5	2825	1691	-564.77	341.53	-563.85	-611.85	0.02	-585.12	0.44
noswot	182	128	5	0	440	30	37	-41.20	5.19	-41	-43.00	0	-43.00	0.02
nsrand-ipx	735	6621	40	1	15	200	17	49233.33	34.01	51200	48880.00	0.04	50217.86	1.56
p2756	755	2756	2	16	0	1492	388	3115.60	173.69	3124	2688.75	0.01	2947.36	0.11
pp08aCUTS	246	240	2	707	387	128	128	7190.34	11.69	7350	5480.61	0	7070.16	0.1
protfold	2112	1835	20	0	16	186	139	-13.00	3471.94	-31	-41.96	0.68	-41.50	4.7
rout	291	556	5	20	12	111	55	1070.23	4.06	1077.56	981.86	0	982.12	0.26
set1ch	492	712	4	506	3640	180	120	52992.08	370.86	54537.8	32007.73	0.01	54038.13	0.1
seymour	4944	1372	64	13	0	83	85	406.86	2820.41	423	403.85	1.09	412.92	8.01
timtab1	171	397	2	23	0	215	91	684878.47	629.81	764772	28694.00	0	466129.32	0.19
timtab2	294	675	3	60	0	253	106	830606.93	1913.24	1096560	83592.00	0	592137.60	0.45
tr12-30	750	1080	6	8	12	192	125	123094.78	1386.79	130596	14210.43	0	129707.41	0.75
vpm2	234	378	2	8	12	191	112	13.36	1.02	13.75	9.89	0	12.86	0.01

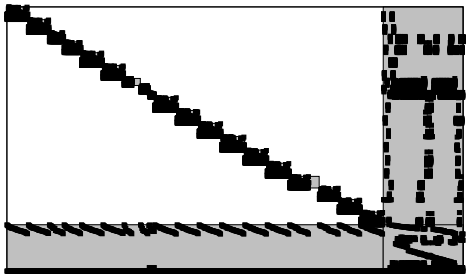
Table 4: Comparison of the quality of dual bounds achievable by our automatic DWR approach as compared to a general-purpose MIP solver for some instances of MIPLIB2003. We are consistently better than the LP relaxation, and sometimes improve on the root node bounds with default cuts enabled. Again, this is a demonstration that improvement is possible, we are not trying to compete in terms of CPU time.



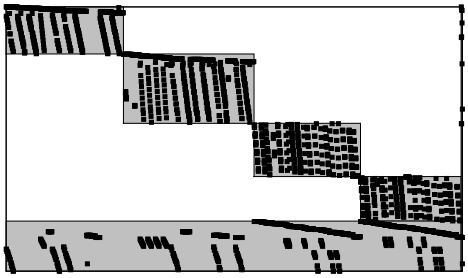
(a) *10teams*



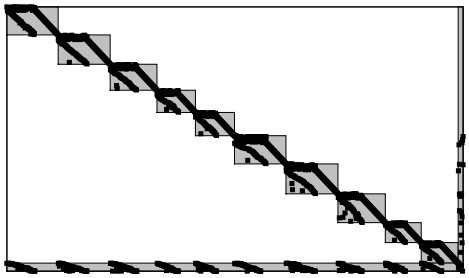
(b) *aflow30a*



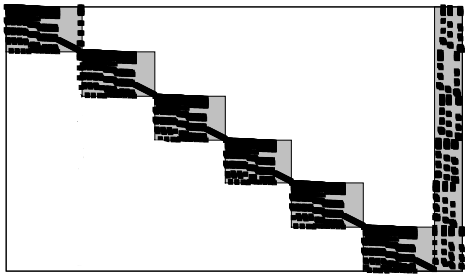
(c) *arki001*



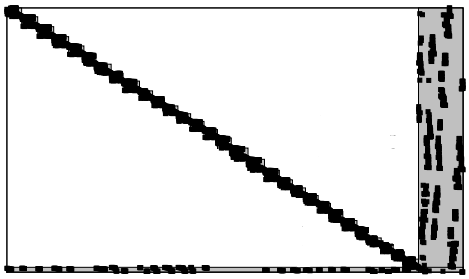
(d) *fiber*



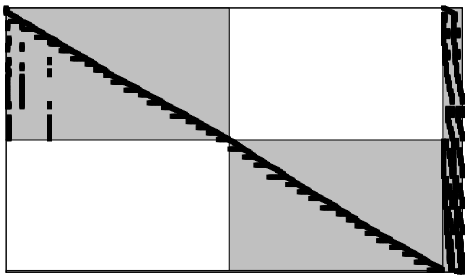
(e) *fixnet6*



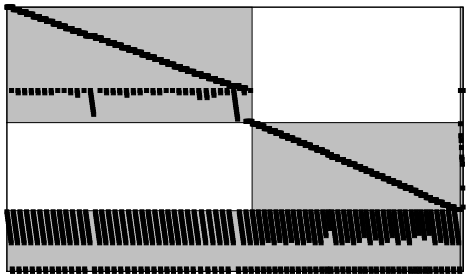
(f) *gesa2*



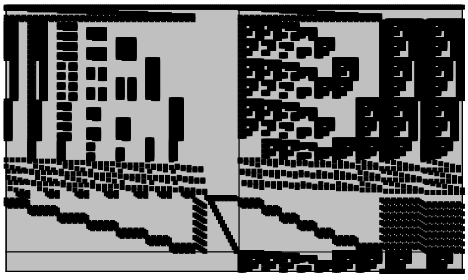
(g) *gesa2-o*



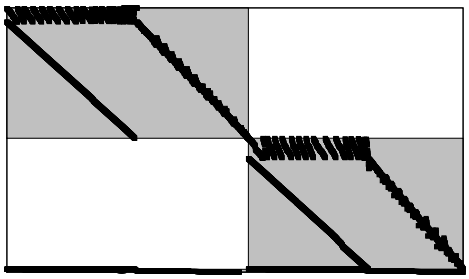
(h) *glass4*



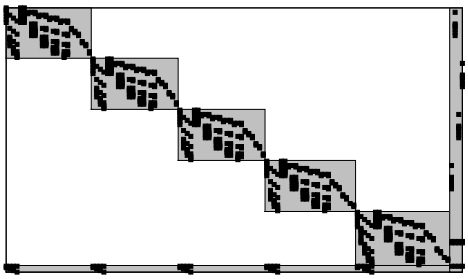
(i) *harp2*



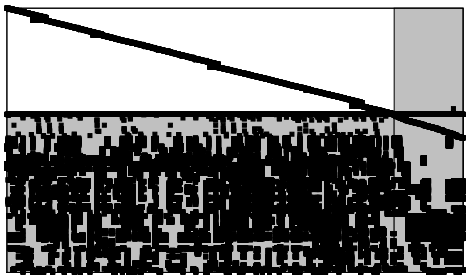
(j) *misc07*



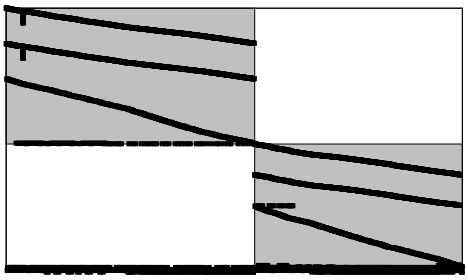
(k) *mks*



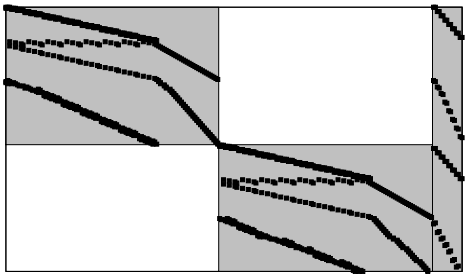
(l) *noswot*



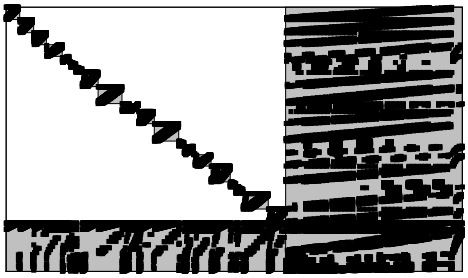
(m) *nsrand-ipx*



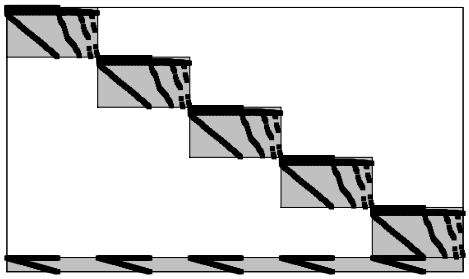
(n) *p2756*



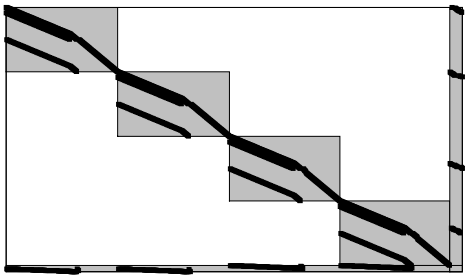
(o) *pp08aCUTS*



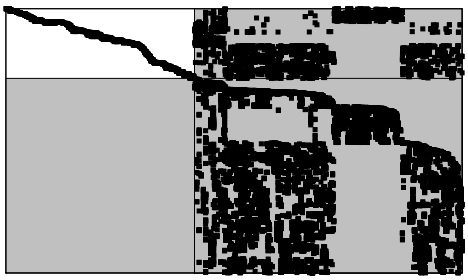
(p) *protfold*



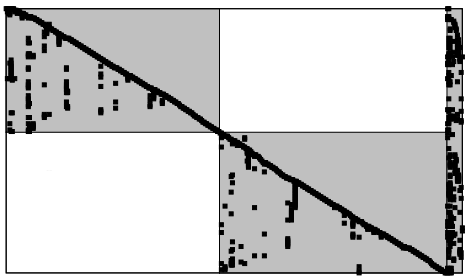
(q) *rout*



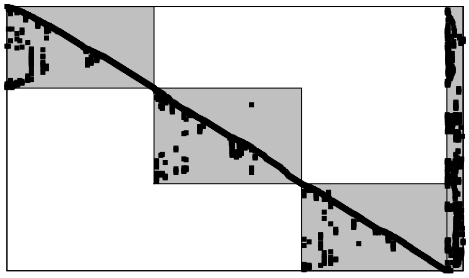
(r) *set1ch*



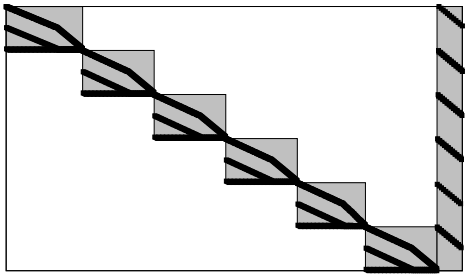
(s) *seymour*



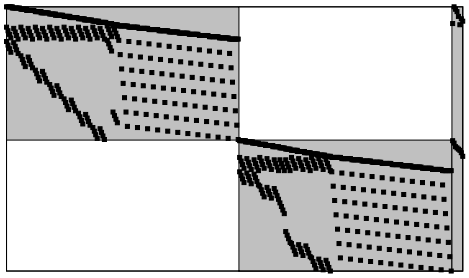
(t) *timtab1*



(u) *timtab2*



(v) *tr12-30*



(w) *vpm2*

Figure 4: Selected arrowhead decompositions for a subset of the MIPLIB2003 instances