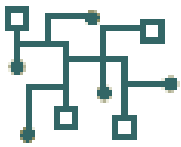

ISE

Industrial and Systems Engineering

Computational Experience with Hypergraph-based Methods for Automatic Decomposition in Discrete Optimization

Jiadong Wang and Ted Ralphs

Department of Industrial and Systems Engineering, Lehigh University, USA

COR@L Technical Report 12T-014

LEHIGH UNIVERSITY

COR@L

COMPUTATIONAL OPTIMIZATION RESEARCH AT LEHIGH

Computational Experience with Hypergraph-based Methods for Automatic Decomposition in Discrete Optimization *

Jiadong Wang[†1] and Ted Ralphs[‡1]

[1]Department of Industrial and Systems Engineering, Lehigh University, USA

November 11, 2012

Abstract

Branch-and-price algorithms based on Dantzig-Wolfe decomposition have shown great success in solving mixed integer linear optimization problems (MILPs) with specific identifiable structure, such as vehicle routing and crew scheduling problems. For unstructured MILPs, the most frequently used methodology is branch-and-cut, which depends on generation of "generic" classes of valid inequalities to strengthen bounds. There has been little investigation into the development of a similar "generic" version of branch-and-price, though this is possible in principle. One of the most important elements required for such a generic branch-and-price algorithm is an automatic method of decomposition. In this paper, we experiment with hypergraph partitioning as a means of performing such automatic decomposition. Computational results explore the potential for applying branch-and-price algorithms within generic solvers and provide insight into how to measure the quality of the decomposition and improve it.

1 Introduction

Branch-and-price methods have been successfully used to solve difficult mixed integer linear optimization problems (MILPs), such as transportation problems in logistic systems, crew scheduling problems in the airline industry, and other large-scale applications [1, 2]. Although branch-and-price methods have the potential to generate strong dual bounds, to reduce symmetry, and to exploit special structures, they generally require that a decomposition of the constraints be either given as input or discovered as part of the solution process. This means that either the user has to have some way of describing the decomposition to the solver or the solver must be capable of discovering that structure automatically.

In this paper, we focus on automatic detection of structure using hypergraph partitioning algorithms. These methods have been widely used for detection of block-diagonal structure (described more formally in Section 2) in the context of certain linear algebraic computations [4], mainly for the purpose of efficient parallelization. In the context of branch-and-price, our goal in exploiting block structure is not only to allow the use of parallel computation but also to improve on

*NSF Grant CMMI-1130914

[†]jiw508@lehigh.edu

[‡]ted@lehigh.edu, coral.ie.lehigh.edu/∼ted

the bounds yielded by solving the linear relaxation, eventually leading to improved solution times overall for certain classes of MILP.

When detecting block structure automatically, it is important to have a measure of "quality" that is easy to compute, since there are usually multiple ways of decomposing a given matrix. When the goal is to parallelize a single matrix computation, the measures of quality are fairly straightforward, but in branch-and-price, measuring the quality of a decomposition is much more difficult. Ultimately, our goal is to reduce the overall computation time as much as possible, but proxies for this measure that are easier to evaluate must be used in practice. One such proxy is the bound improvement achieved in the root node, which also has the advantage of being unaffected by changes to other algorithmic strategies, such as branching. Unfortunately, the computation of this quantity is itself too expensive in general, so one of the goals of this study is to assess cheaper alternatives.

The work reported on herein follows lines of development similar to those of the study reported in [3], which was performed in support of development of the GCG framework [9]. GCG is a solver built on top of the constraint integer programming framework SCIP that implements an approach to generic Dantzig-Wolfe decomposition similar to the DIP framework that is employed in this study [7]. Both this study and the aforementioned one develop methodologies for detecting matrix structure by using hypergraph partitioning algorithms. Both studies also propose measures of goodness for the quality of a given decomposition. The present study differs by focusing on producing singly-bordered block-diagonal matrices, as opposed to the doubly-bordered structure favored in [3], which requires a different hypergraph partitioning model. Furthermore, our measures of goodness take into account not only the distribution of nonzeros, but also the distribution of nonzeros in columns corresponding to integer variables. Finally, we also report on a number of issues that arise in tuning the hypergraph partitioning software. As far as we know, these are the only existing studies focused on automatically identifying structure for the purposes of performing a Dantzig-Wolfe decomposition.

The remainder of the paper is organized as follows. Section 2 reviews the generic implementation of the branch-and-price algorithm that is the focus of this paper. Section 3 describes the hypergraph models we use to achieve matrix structure detection. Section 4 presents the computational results from the MIPLIB instances and provides insight into how to measure the quality of automatic detected structure. Section 5 summarizes the paper and points out the future work in this direction.

2 Generic Branch and Price

We consider solution of an MILP, which is to compute

$$z_{IP} = \min\{c^\top x \mid Ax \leq b, x \in \mathbb{Z}^r \times \mathbb{R}^{n-r}\}, \tag{1}$$

where $A \in \mathbb{Q}^{m \times n}$, $c \in \mathbb{Q}^n$, and $b \in \mathbb{Q}^m$. A *decomposition* is a partition of the rows of $[A, b]$ into two sub-matrices $[A', b']$ and $[A'', b'']$. The decomposition is chosen such that the solution of the relaxation obtained by dropping the linear constraints represented by the sub-matrix $[A'', b'']$ is practical (relative to solution of the original problem). The principle underlying decomposition methods is to exploit the solvability of this relaxation in order to solve the original problem more efficiently.

A central challenge is to find an effective decomposition. Typically, this process is a manual trial-and-error process. Here, we describe a method for automating it. For convenient reference throughout, we denote

$$\mathcal{Q}' = \{x \in \mathbb{R}^n \mid A'x \leq b'\}, \tag{2}$$
$$\mathcal{Q}'' = \{x \in \mathbb{R}^n \mid A''x \leq b''\}, \text{ and} \tag{3}$$
$$\mathcal{P}' = \text{conv}(\mathcal{Q}'). \tag{4}$$

For ease of exposition, we assume that $\mathcal{Q}'$ and $\mathcal{Q}''$ are bounded. Most solution methods for MILP are based on the branch-and-bound algorithm and require a method for efficiently computing a lower bound on z_{IP}. The most obvious way of obtaining such a bound is by relaxing the integrality requirements on the variables to obtain

$$z_{LP} = \min_{x \in \mathcal{Q}'' \cap \mathcal{Q}''} c^\top x. \tag{5}$$

This linear optimization problem (LP), referred to as the *linear relaxation*, can be solved efficiently and results in what we refer to as the *LP bound*. A potentially stronger bound, which we call the *Dantzig-Wolfe bound*, is

$$z_{DW} = \min_{x \in \mathcal{P}' \cap \mathcal{Q}''} c^\top x. \tag{6}$$

Since $\mathcal{P}' \subseteq \mathcal{Q}'$, we have that $z_{DW} \geq z_{LP}$. Furthermore, since $\mathcal{P}'$ is bounded, the representation theorem allows us to write

$$\mathcal{P}' = \{x \in \mathbb{R}^n \mid x = \sum_{p \in \mathcal{E}} p\lambda_p, \sum_{p \in \mathcal{E}} \lambda_p = 1, \lambda_p \geq 0 \ \ \forall p \in \mathcal{E}\}, \tag{7}$$

where $\mathcal{E}$ is the set of extreme points of $\mathcal{P}'$. Using this representation, we can re-write (6) in the explicit form

$$z_{DW} = \min_{\lambda \in \mathbb{R}^{\mathcal{E}}_+} \{c(\sum_{p \in \mathcal{E}} p\lambda_p) \mid A''(\sum_{p \in \mathcal{E}} p\lambda_p) \geq b'', \sum_{p \in \mathcal{E}} \lambda_p = 1, \lambda_p \geq 0 \ \ \forall p \in \mathcal{E}\}, \tag{8}$$

which we refer to as the *master problem*.

Generating the set $\mathcal{E}$ is usually impractical, so we use a column generation procedure to dynamically generate members of $\mathcal{E}$, as shown in Figure 1. The first step is to generate an initial subset $\mathcal{E}^0$ of $\mathcal{E}$, which is augmented in each subsequent iteration. In iteration t, we solve (8) with $\mathcal{E}$ replaced by $\mathcal{E}^t$ (we refer to this problem as the *restricted master problem* (RMP)) to obtain the dual solution (u^t, α^t). The *column generation subproblem* is to generate new members of $\mathcal{E}$ whose reduced costs in the RMP with respect to the current dual solution are negative. We iterate until no more columns with negative reduced cost can be generated, at which time, we have that $z^t_{DW} = z_{DW}$ (in principle). For more details on the procedure, see [5].

The branch-and-price algorithm is a branch-and-bound algorithm in which the bounds at each node is obtained by this procedure. Due to its success in different large-scale optimization applications, a number of software frameworks have been developed to aid in the implementation of branch-and-price algorithms, including BaPCod [6], DIP [7], GCG [9], and ABACUS [8].

Dantzig-Wolfe Method

1. **Initialize**: Construct an initial approximation from an initial set $\mathcal{E}^0$ of extreme points of $\mathcal{P}'$.

2. **Master Problem:** Solve the Dantzig-Wolfe reformulation

$$\bar{z}^t_{DW} = \min_{\lambda \in \mathbb{R}^{\mathcal{E}}_+} \{c(\sum_{p \in \mathcal{E}} p\lambda_p) \mid A''(\sum_{p \in \mathcal{E}} p\lambda_p) \geq b'', \sum_{p \in \mathcal{E}} \lambda_p = 1, \lambda_p \geq 0\}$$

   to obtain the optimal value $\bar{z}^t_{DW} = \min_{\mathcal{P}^t \cap \mathcal{Q}''} cx \geq z_{DW}$, an optimal primal solution λ^t, and an optimal dual solution (u^t, α^t).

3. **Subproblem:** Solve the subproblem $\min_{x \in \mathcal{P}'} \{(c - u^t A'')x - \alpha^t\}$ to generate a set $\tilde{\mathcal{E}}$ of improving members of $\mathcal{E}$ (those with a negative objective function values).

4. **Update:** If $\mathcal{E} \neq \emptyset$, set $\mathcal{E}^{t+1} \leftarrow \mathcal{E}^t \cup \tilde{\mathcal{E}}$ to form the new approximation

$$\mathcal{P}^{t+1} = \{ \sum_{p \in \mathcal{E}^{t+1}} p\lambda_p \mid \sum_{p \in \mathcal{E}^{t+1}} \lambda_p = 1, \lambda_p \geq 0\}$$

   and set $t \leftarrow t+1$. Go to Step 2.

5. If $\tilde{\mathcal{E}} = \emptyset$, output the bound $z_{DW} = \bar{z}^t_{DW} = \underline{z}^t_{DW}$.

Figure 1: Outline of the Dantzig-Wolfe method

In contrast to the well-known branch-and-cut method, it has proven difficult in general to develop a "generic" version of branch and price, requiring no input from the user beyond the model itself. This is because most frameworks leave both the method for finding the decomposition and the method for solving the column generation subproblems unspecified. A generic variant of the procedure in Figure 1 can be obtained, however, by (1) using automatic methods of identifying block structure in the matrix to produce candidates for the decomposition, (2) solving the column generation subproblem using a generic MILP solver, and (3) using a generic method of branching on variables in the original problem formulation. When employed within a branch-and-bound algorithm, we obtain a variant of the branch-and-price algorithm we refer to as *generic branch-and-price*.

The main idea of automatic decomposition is to identify a permutation of the rows and columns of A that exhibits so-called block-diagonal structure in the matrix A, as illustrated in Figure 2. Here, the matrix A' is comprised of k disjoint blocks of nonzero elements. When A' has this structure, the column generation subproblem decomposes naturally into k independent (and much smaller) MILPs. This is the property we wish to exploit and the motivation for identifying this structure.

The nonzeros of A'' can appear in any column and the corresponding constraints serve to link the k disjoint sets of constraints of $[A', b']$. These are generally referred to as the *coupling constraints*.

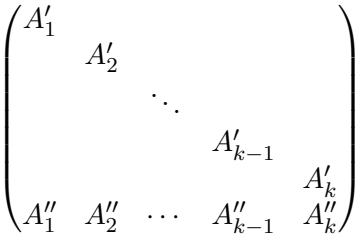

Figure 2: Singly-bordered block-diagonal matrices

The structure shown in Figure 2, called a *singly-bordered* block-diagonal matrix. It is also possible to have a set of columns that are allowed to have nonzeros in any row, in which case we have a *doubly-bordered* structure. We restrict the discussion here to the singly-bordered case. For details on the doubly-bordered case in this setting, see [3].

3 Hypergraph-based Partitioning Methods

A number of methods to identify the block structure of matrices have been proposed. Martin et al. developed an integer optimization model for identifying the block structure of a matrix [10]. The objective in their model was to minimize the number of coupling rows, but they included constraints to ensure that the decomposed matrix should tend to be "balanced" by placing an upper bound on the number of rows in each block. This problem was shown to be NP-hard and even with the development of a specialized branch-and-cut algorithm, solving it as a preprocessing step within an algorithm for generic MILP proved not to be practical.

Graph and hypergraph partitioning provide a more computationally practical heuristic approach to automatic decomposition. Ferris and Horn describe heuristics for detecting matrix structure using traditional graph partitioning in order to solve linear optimization problems in parallel [11]. Since there are drawbacks to graph partitioning methods (see [13] for details), we focus here instead on hypergraph partitioning to detect the underlying block structure. A hypergraph is a generalization of a traditional graph, in which each edge (called a *hyperedge* or a *net* in hypergraph terms) consists of a subset of nodes of arbitrary cardinality (in contrast to a traditional graph in which edges are subsets of cardinality two). A hypergraph is thus defined as $\mathcal{H} = (V, E)$, where V is the set of nodes (vertices) and $E \subseteq 2^V$ is the set of nets (hyperedges).

Given a vector of weights $w \in \mathbb{R}^E$, the K-way hypergraph partitioning problem is to partition the set of nodes of a hypergraph $\mathcal{H}$ into at most K subsets while minimizing the weight of the resulting *cut*, which consists of the sum of the weights of all edges having a nonempty intersection with more than one member of the partition. A secondary objective, usually expressed as a constraint, is for the cardinalities of the members of the resulting partition to be approximately the same, i.e., for the partition to be *balanced*. If the nodes also have weights, we may instead want the sum of the weights of the nodes in each member of the partition to be balanced. Currently, so-called multilevel heuristics are the most successful approaches to performing both K-way graph and hypergraph partitioning.

To use a hypergraph partitioning algorithm to find (singly-bordered) block-diagonal structure

in a matrix, we use a *row-net* model in which we identify each column of the matrix with a node in an associated hypergraph and identify each row of the matrix with a hyperedge consisting of the nodes associated with the columns in which the row has nonzero elements [12]. After mapping the matrix to a hypergraph in this way and finding a partition of the hypergraph, we can identify the structure of the original matrix as follows. The rows of each block consist of the set hyperedges whose elements are completely contained in one of the partitions of the node set. The coupling rows consist of all hyperedges in the cut, i.e., hyperedges having nonempty intersection with more than one element of the partition. Thus, minimizing the size of the cut is the same as minimizing the number of coupling rows (when the edges have unit weights). The balance constraint can be interpreted as ensuring that the blocks have approximately equal numbers of columns (again, assuming unit weights).

Figure 3 gives an example, showing the decomposition of the constraint matrix of a MIPLIB2003 instance *a1c1s1* in singly-bordered diagonal form. The left figure shows the original constraint matrix of *a1c1s1*, which does not expose a block structure. By a 6-way partitioning based on the row-net hypergraph model, the block structure is detected as shown in the right figure.

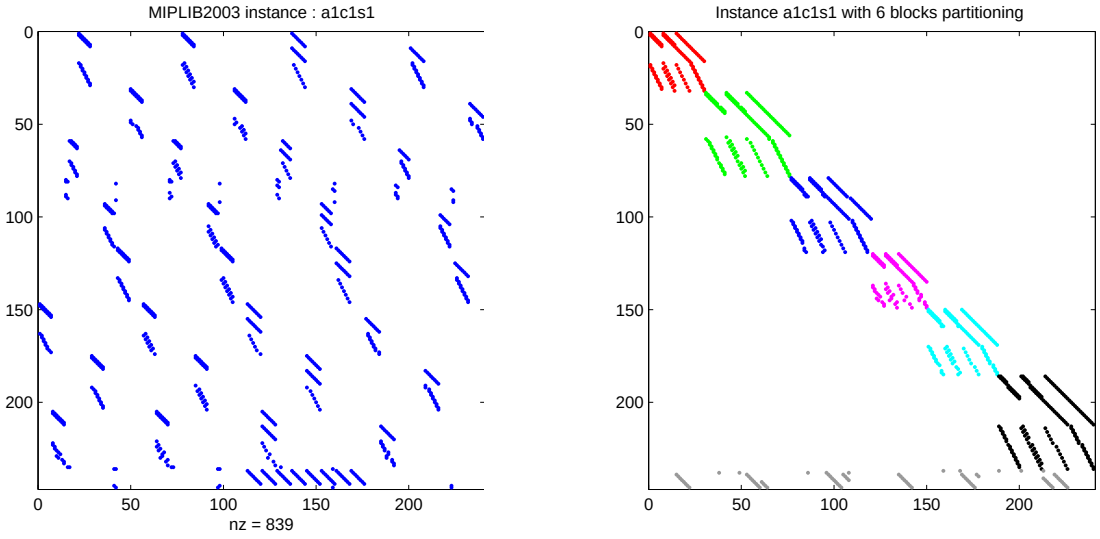

Figure 3: Decomposing a MIPLIB2003 instance *a1c1s1* into a singly-bordered diagonal matrix by a 6-Way hypergraph partitioning

4 Computational Experiments

The computational study here serves three purposes. The first is simply to explore the promise of generic branch-and-price as a potential alternative to more traditional branch-and-cut algorithms for solving generic MILP. The second is to provide some insight into how to compare and tune the available hypergraph partitioning procedures in order to produce decompositions of high quality. The third and perhaps the most challenging goal is to provide insight into how we can effectively measure the quality of a decomposition to begin with. Without effective quality measures, the first two goals are not achievable. We want to emphasize that the results reported here are preliminary

in nature and should be considered the first steps in what we hope will be a long line of future studies. While we do think that our results indicate promise, there is much work to be done and the tests are necessarily limited in scope. The small set of MILPs on which the experiment here should not be considered "representative." Our overarching goal is to motivate further study and establish a general framework and direction for that study.

The experiments here were performed on compute nodes with dual eight core 0.8GHz AMD Opteron(tm) processors having 32G memory and 512KB cache per core. We used the hMETIS and PaToH hypergraph partitioning tools [16, 17] to detect the block structure and the DIP (Decomposition in Integer Programming) framework, an open-source software that is part of the COIN-OR repository, to implement the resulting generic branch-and price algorithm [7]. We used the open-source linear optimization problem solver CLP [14] to solve the restricted master problem and CBC [15] to solve the subproblem. The test instances were selected from MIPLIB2003 and MIPLIB2010 from among instances of medium size. The computational time limit was 2 hours in all cases. When exceeding the cutoff, the lower bound reported is the best one attained until that time.

Although we are ultimately concerned with overall solution times, we focus in this study on the optimality gap in the root node, which is an often-used measure of goodness for methodologies based on branch and bound. This measure is imperfect at predicting impact on solution time, but it is nevertheless a good starting point for comparing solution methodologies. For the purposes of comparing branch and price with branch-and-cut, we compare DIP with CBC in terms of bound quality at the root node. To calculate the optimality gap, the optimal solution is used as as baseline (as opposed to the solution produced in the root node). The optimality gap closed is the improvement of either the Dantzig-Wolfe bound or the bound calculated by CBC in the root node over the LP bound.

4.1 Comparing PaToH and hMETIS

PaToH and hMETIS are both well-known hypergraph partitioners and can be used effectively to discover block structure. Here, we compare these two partitioners with respect to our current purpose. To compare the two methods under typical use, Çatalyürek, the developer of PaToH, used 134 hypergraphs arising from different areas such as sparse matrix-vector multiplication, linear optimization, and very large-scale integrated circuit design problems as benchmarks to compare these two partitioners [19]. The results show that PaToH is significantly faster than hMETIS with default settings and the difference in execution time increases as problem size increases. However, the partition quality both in terms of imbalance and weight of the cut favors hMETIS, which shows the expected trade-off between quality and speed.

In the context of exploiting block-diagonal structure in MILPs, the trade-off between quality and speed is quite different. First, the quality measure is no longer the cut size but rather the Dantzig-Wolfe bound and ultimately the overall computational time to solve the optimization problem. Second, due to the smaller size of the hypergraphs arising in optimization problems as compared to those arising in other domains, as well as the fact that the execution time for exploiting the block structure is small relative to the total time required for solving an MILP, the speed of matrix block partitioning is not of primary concern here. Thus, the comparison between these two hypergraph partitioner is based on the root node bound.

Some results of the comparisons are shown in Table 1. Although the maximum number of blocks is an input that can be varied in general (see Section 4.3 for a discussion of this), we fixed

Table 1: Comparison of hMETIS and PaToH

instance	cols	rows	opt	LP	hMETIS DW bound	Block Num	PaToH DW bound	Block Num
noswot	128	182	-41	-43	-42.2	3	-41	1
10teams	2025	230	924	917	924	3	924	3
pp08a	240	136	7350	2748	7168	3	6985	2
timtab1	397	171	764772	286940	523871	3	365619	3
vpm2	378	234	13.7	9.889	12.734	3	11.9	3
k16x240	480	256	10674	2769.838	3268.9	3	5849	2
fiber	1298	2944	405935.18	156082.5	156082.5	3	405935.18	2
pg	2700	125	-8674.3426	-11824.657	-9667.97651	3	-8692.8	2
set1ch	712	492	54537.75	32007.7299	54517.67649	3	54335.112	2
p80x400b	800	480	39667	6418.8	37650.21	3	37517.48	3
fixnet6	878	478	3983	1200	3230.911	3	3229.58	3
vpm2	378	234	13.75	9.889	12.07	3	12.98	2
ran14x18	504	284	3712	3016.944	3160.98	3	3167.524	2
ran16x16	512	288	3823	3116.429	3324.06	3	3369.73757	2
aflow30a	792	552	80598.43	29624.69	74585.46	3	77843.97	2

the maximum number of desired blocks at three for the purposes of comparison. The actual number of blocks output could be less, though this would indicate an inability of the solver to construct a "well-balanced" partition, since fewer blocks are produced when one element of the partition is empty (indicating a solution that is not balanced).

Below are a few insights obtained from the computational experiments

- hMETIS produces more balanced solutions in general and the number of blocks produced is typically equal to the given maximum.
- There are cases for which PaToH produces fewer than the given maximum number of blocks, in which cases the bound from PaToH is often better, since fewer blocks means fewer, larger subproblems that yield stronger bounds in general (but are also more difficult to solve). In our experiments, quite a few instances were partitioned into 2 blocks while our desired number of blocks was 3 for PaToH.
- Generally, when the number blocks produced is the same, hMETIS performs better than PaToH in terms of bound improvement at the root node.
- When using PaToH, one might set the maximum number of blocks higher than what is desired in order to avoid the situation described above. Also, it is suggested to have dummy isolated nodes in the hypergraph representation of matrix for the same reason.

Based on these observations, we used hMETIS for the experiments in the remainder of the paper.

4.2 Time for block structure detection

One of the most important considerations for any block structure detection method is the time required to do the partitioning. The computational time to perform the hypergraph partitioning

for instances in Table 3 is generally less than one second. Given the expected solution time for these instances, we regard this as negligible. Even for instance with up to tens of thousands variables, the partitioning procedure takes at most a few seconds.

4.3 Parameter tuning

Both hMETIS and PaToH use multilevel heuristic methods consisting of three phases: a coarsening phase, an initial partitioning phase, and an uncoarsening phase [18]. Before executing the partitioning procedure, there are two primary parameters that need to be set, both of which may affect the resulting bound improvement and overall decomposition quality. One is the maximum number of elements in the partition (number of blocks in the resulting partitioned matrix) and the other is the weights of the nodes and hyperedges. The experiments below illustrate the impacts of variation of these parameters on the decomposition.

Number of blocks. To illustrate what happens when the number of blocks is varied, the four subfigures in Figure 4 provide a visualization of block structure with different numbers of blocks for a single instance. It is difficult to tell from these figures which of these decompositions would be the most effective in our present setting and we have found this to be true in general. Preliminary experiments failed to find a strong correlation between the number of blocks and the bound achieved, though clearly the number of blocks must have some impact, since we know that in the extreme cases of a single block, we obtain the strongest possible bound, whereas many extremely small blocks will clearly yield a weak bound. Figure 5 shows the relationship between the optimality gap closed and the number of blocks on several instances from MIPLIB. For instance *go*19, the optimality gap closed decreases as the number of blocks increases, as perhaps would be expected. Not too surprisingly, there are no cases in which the bound increases reliably as the number of blocks increases. Interestingly, however, there are cases, such as the instance *swath*, for which the optimality gap increases and then decreases as the number of blocks increases.

To further illustrate this phenomenon, we take computational time into consideration on some selected instances. Figure 6 shows the relationship between the Dantzig-Wolfe bound, the number of blocks, and the computational time required to obtain the bound. Generally, it can be observed that as the number of blocks increases, the time required decreases, since the individual blocks are getting smaller and the subproblems are easier to solve. For the instance *pg*, the "optimal" block number can be seen to be eight, since the both Dantzig-Wolfe bound and computational time benefit from the higher number of blocks. For instance *machrophage*, however, there is a trade-off between the bound improvement and the overall computation time, since the bound gets worse and the computational time decreases as we increase the number of blocks. The "optimal" block number is four if we consider these two factors.

The question remains how to choose the number of block a priori. The question of how to do this dynamically is very difficult and we do not attempt to investigate it further in this paper. We point out, however, that one obvious strategy is to construct different partitions on each thread of a multi-core computer in parallel and compare their quality based on the measures introduced below. At this time, it appears difficult to select the number of blocks in an intelligent fashion based on other factors.

Weight of nodes and hyperedges. Another important parameter in hypergraph partitioning is the weights chosen for the nodes and hyperedges, which control the balance and other properties

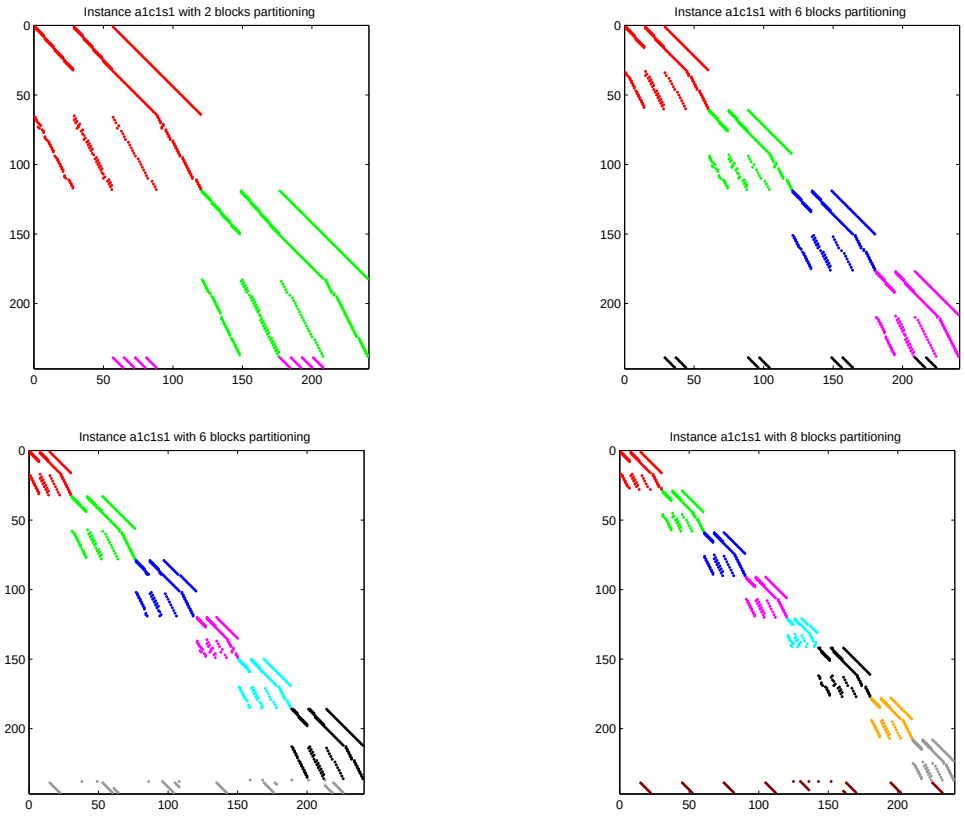

Figure 4: The detected structure with 2, 4, 6, and 8 blocks

of the decomposition. In hypergraph partitioning, keeping balance means to keep the sum of the weights of the nodes in each partition approximately equal. In general, our expectation is that decompositions in which the integer variables play a strong role in the individual blocks (i.e., the constraints of the blocks involve as many of the integer variables as possible) will yield strong bounds, since if none of the blocks have integer elements (the nonzero variable which is constrained to be integer in the original optimization problem), the Dantzig-Wolfe bound is known to be equal to the LP bound.

One strategy, therefore, is to assign a higher weight to nodes corresponding to integer variables and to hyperedges that include a high number of such nodes. Table 2 below compares the Dantzig-Wolfe bound obtained with unit weighted (left column) and non-unit weighted (right column) nodes and hyperedges. For non-unit weighted cases, we assigned a weight of two to integer variables and a weight of one to continuous variables. Weights of hyperedges were assigned to be the total number of nonzero elements in their corresponding rows. We observe that out of 25 instances, 6 instances show a decrease with weighting and the rest show a bound at least a high as in the unit weight case. Eleven instances show a strict bound increase. It thus appears that adjusting the weight of nodes and hyperedges intentionally should help in most cases and this agrees with our intuition.

Table 2: Comparison of unit weighted and non-unit weighted hypergraphs

instance	unit weighting	non-unit weighting
10teams	924	924
fiber	393232	396716.3
p2756	3119	3119
noswot	-43	-43
pp08a	7139	7104.3
pp08aCUTS	7104.3	7144.8
mkc	-565	-564.7
modglob	20670945.9	20671798.8
gesa2	25779856.3	25782225.3
bienst2	37.5	31.5
enlight13	13	12
rout	1031.82278	1070.2
set1ch	54517.86	54517.86
pg	-8696.89	-8703.8
qiu	-931.63	-931.63
pg5_34	-14443.3	-14406.0
pw_myciel4	4	4
p80x400b	37385.5	37599.5
opt1217	-20.02	-20.02
ns894788	7	6.2
mik-250-1-100-1	-70419.5	-70419.5
macrophage	369	370
fixnet6	3230.9	3229.5
csched010	364.3	368.1
beasleyC3	564.1	586.28387

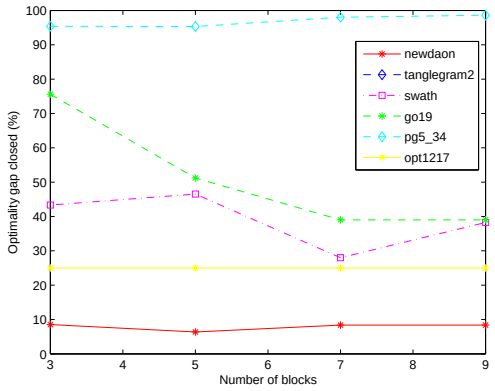

Figure 5: Relationship between number of blocks and optimality gap closed.

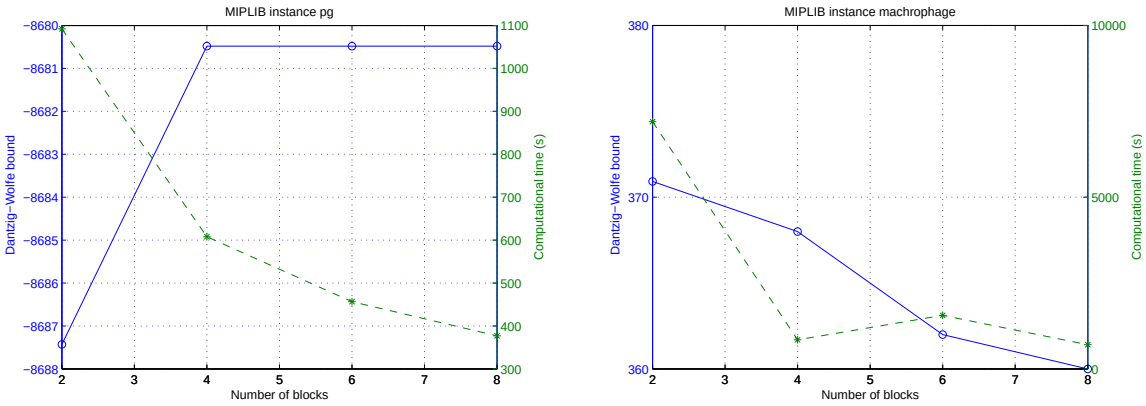

Figure 6: Relationship between number of blocks, Dantzig-Wolfe bound, and computation time.

4.4 Root bound

Here, we explore how the bound achieved with automatic decomposition compares to the bound that can be achieved with cutting planes in the root node, as described earlier. Note that there is an obvious potential trade-off between the computation time and the root bound achieved that can be examined by varying the number of blocks. In this paper, we did not experiment with methods of choosing the number of blocks automatically and have instead fixed the number of blocks at three, since this seems likely to yield good bounds on a large spectrum of instances and provides a good trade-off with computation time. While setting the weight of nodes and hyperedges can improve the bound for most of the instances, as mentioned above, the experiments here are based on the unit weight of node and hyperedges, which do not involve any specific tuning and still result in a strong bound in many cases.

The results of our experiments are shown in Table 3. The columns labeled *DW %*, *LP %*, and *C %* indicate the optimality gaps achieved with the Dantzig-Wolfe relaxation, the linear relaxation, and the cutting plane method, respectively, while the columns labeled *close %* are the percentage of the LP gap closed by the respective method. Overall, the results show that generic branch and

price can provide strong bounds at the root node, sometimes even stronger than that obtained by the cutting plane method (results shown in bold). Of course, we must point out that the same cutting planes are used to solve the subproblems in the Dantzig-Wolfe implementation as are used in CBC's branch and cut. Nevertheless, the comparison provides an interesting starting point for further investigation. Note that the results shown in this table were obtained using hMETIS.

Table 3: Results of experiments

instance	cols	rows	k	LP %	DW %	close %	C%	close %
10teams	2025	230	2	0.76	0	100	0	100
fiber	1298	363	3	61.5	3.1	95	2.4	96
p2756	2756	755	2	13.9	0.1	99	0.2	98
noswot	128	182	3	4.88	2.9	40	4.88	0
pp08a	240	136	3	62.6	2.87	95.4	5.8	90.7
pp08aCUTS	240	246	3	25.43	3.3	87	7.4	70
mkc	5325	3411	3	8.5	0.2	97.6	3.2	62.3
modglob	422	291	3	1.5	0.3	79.9	0.2	86.6
gesa2	1224	1248	3	1.1	10e-8	100	0.02	98.2
bienst2	505	576	3	78.5	31.3	60.3	26	66.8
enlight13	383	169	3	100	81.6	18.4	92.4	7.59
rout	556	291	3	8.88	4.24	52.25	8.4	5.4
set1ch	712	492	3	41.3	0.036	99.9	1.8	95.6
pg	2700	125	3	36.3	0.26	99.28	0.07	99.8
qiu	840	1192	2	601	601	0	542.7	9.7
pg5_34	2600	225	3	16	0.7	95.6	0.19	98.8
pw_myciel4	1059	8164	3	100	60	40	60	40
p80x400b	800	480	3	83.8	5.75	93.1	20.8	75.1
opt1217	769	64	3	25.1	25.1	0	10.6	57.7
ns894788	3463	2279	3	10	0	100	0	100
mik-250-1-100-1	251	151	3	19.6	5.5	72.3	3.6	82
macrophage	2260	3164	3	100	1.3	98.7	18.9	81
fixnet6	878	478	3	69.8	18.8	73	7.7	88.9
csched010	1758	351	2	18.5	10.7	42.1	14.1	23.7
beasleyC3	2500	1750	3	94.6	25.1	73.4	19.8	79

4.5 Quality measures for decomposition

The core obstacle in automating all aspects of the branch-and-price algorithm is to understand when the branch-and-price algorithm will work well for generic MILP problems. In the context of automatic block structure detection, we need to measure the quality of the decomposition. As is traditional in machine learning, we refer to numeric quantities that may indicate the quality of a decomposition as *features* and propose some here. Finally, we combine these measures into a single quality metric. Here are the five features we considered.

- Fraction of nonzero elements in the coupling rows (α) (ratio of the number of nonzero elements in the coupling rows and the total number of elements in the coupling rows).

- Fraction of elements in the coupling rows that are integer (β) (ratio of the number of integer elements in the coupling rows and the total number of nonzero elements in the coupling rows).
- Fraction of integer elements in the coupling rows (γ) (ratio of the number of integer elements in the coupling rows and the total number of integer elements in the matrix).
- The mean fraction of integer elements in the blocks (η) (the average mean value of the ratio of the number of integer elements in a given block and the the number of nonzero elements in that block, across all blocks).
- The sample standard deviation of the measure η (θ).

Our initial conjecture is that lower values of α, β, γ, and $1 - \eta$ will lead to higher quality decompositions (as indicated by root bound), since it seems natural we would want more rows and nonzeros (both overall and in the columns corresponding to integer variables in particular) in the blocks. Motivated by the success of practical application in which branch-and-price algorithms were used, we also conjectured that achieving balance in the number of rows and the number of integer elements in each block should yield good results. Here, we use θ as a measure of the balance among the blocks.

Figure 7 shows the relationship between the five potential measures and the DW optimality gap closed. Two out of the five measures (α, γ) appear to have a correlation with optimality gap closed and DW bound improvement, although there are outliers and the relationship is not very strong. Features β and η also appear to show a relationship, but it is not consistent enough for a wide range of instances. The standard deviation feature θ also does not clearly show potential.

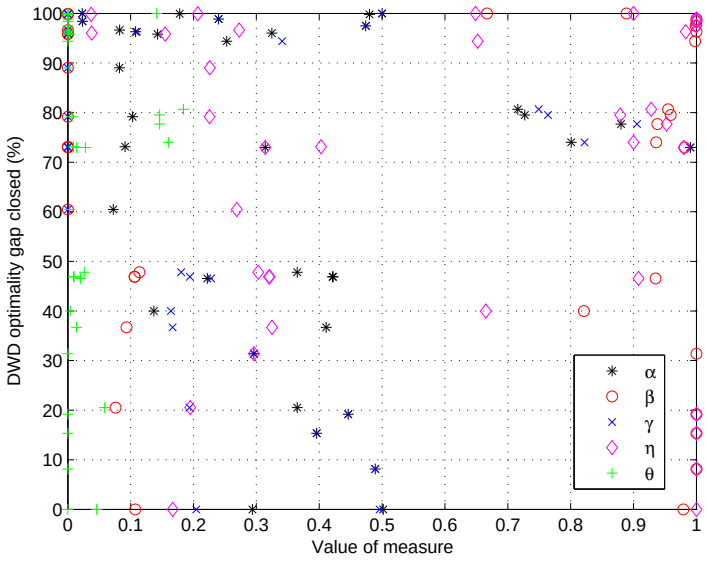

Figure 7: The relationship between potential features and optimality gap closed

Based on the relationship between α, γ and bound improvement, we propose a prediction measure to estimate the quality of detected structure at the root node. It is a function of α and γ,

expressed as

$$\Pi = 1 - \min(\alpha, \gamma)),$$

For this test set, the measure Π, which has a value between 0 and 1, appears to have a positive correlation with optimality gap closed. Tests on 20 new instances from MIPLIB shown in the Figure 8 show the positive linear relationship. The Figure 8 also shows a possible linear relationship between the metric and the optimality gap that can be closed. While these results are promising, we repeat our earlier caveat that much work remains to be done and these conclusions should be considered preliminary.

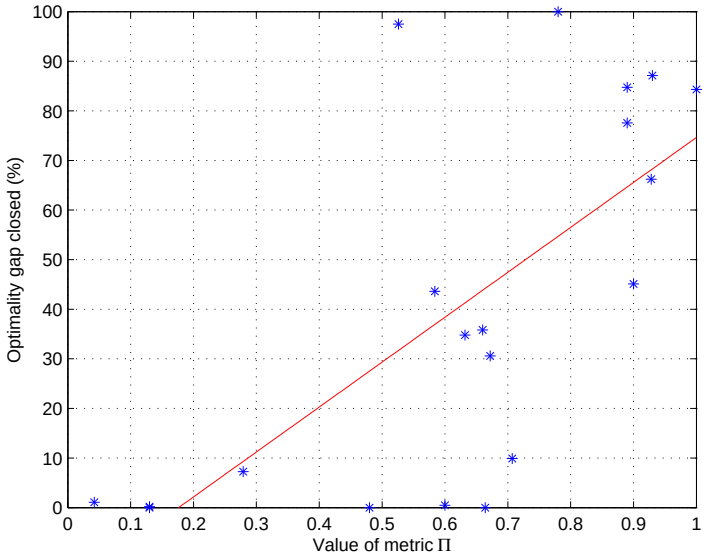

Figure 8: The relationship between optimality gap closed and proposed metric

5 Conclusions

In this paper, we discussed the use of the row-net hypergraph model to detect block-diagonal structure of generic MILP problems automatically. With respect to bound at the root node, automatic decomposition shows promise, but for these methods to be truly practical, computation time must be taken into account. The final goal is not to replace cutting plane methods, but rather to supplement them. By honing our ability to determine in a preprocessing step when decomposition might be effective, we hope to develop solvers capable of switching between cutting plane and column generation methods as appropriate. Of course, there has also been substantial work on methods of generating cutting planes *within* a branch-and-price framework. Much work remains to be done in investigating these so-called *branch-and-cut-and-price* methods.

Moving forward, there are many obvious future research directions to pursue and we mention a few here.

- This paper focuses on singly-bordered block-diagonal structure detection, but it may be advantageous to consider a more general framework allowing for doubly-bordered structure, as in [3].

- There is more work to be done in determining how to judge the quality of a given decomposition. Primarily, we need to take computation time into account as a component of goodness rather than simply considering the root bound, which can be easily manipulated. In this regard, there may be additional features that relate to decomposition quality that have not yet been explored. Even with an explicit metric, choosing the numeric threshold for determining whether the decomposition is good or not is difficult. Motivated by recent machine learning techniques applied in tuning and preprocessing the optimization solvers, we may use support vector machine (SVM) to classify the given decompositions according to their suitability for computation. Existing instances about which we already have deep knowledge can be used for training.

- In this study, we did not consider methods of choosing the number of blocks automatically based on properties of a specific instance. In many cases, the matrix may have a natural block structure to being with and our methodology should be capable of detecting that and respecting this natural number of blocks. Otherwise, we may actually end up destroying existing structure with our automatic methods. As a next step, we would also like to take into account the possibility of solving the subproblems in parallel by exploiting the block structure. In this case, we should take into account the number of available cores/processors when choosing the number of blocks.

- Finally, we have yet to experiment fully with how we can use the variance of node and edge weights to improve the quality of the decomposition. At the moment, we are using the partitioning software "out of the box," but it is our long-term goal to develop specialized methods for achieving decompositions that will be effective in the specific context discussed here.

Much work remains to be done and the potential for these methods to work in a completely generic fashion is still unclear. We are convinced that they do have a role to play and hope that this work will encourage more investigation.

References

[1] Barnhart C., Johnson E., Nemhauser G.L., Savelsbergh M.W.P., Vance P.H. 1998. Branch-and-Price: Column Generation for Solving Huge Integer Programs. 1998. *Operations Research.* **46**(3) 316-329.

[2] Wilhelm W.E. 2001. A Technical Review of Column Generation in Integer Programming. *Optimization and Engineering.* **2**(2), 159-200.

[3] Bergner M., Caprara A., Ceselli A., Furini F., Lübbecke M.E., Malaguti E., Traversi E.. Automatic Dantzig-Wolfe Reformulation of Mixed integer Programs. Available: `http://www.optimization-online.org/DB\_FILE/2012/09/3614.pdf`

[4] Çatalyürek, Ü., Aykanat. C. Hypergraph-Partitioning Based Decomposition for Parallel Sparse-Matrix Vector Multiplication. 1999. *IEEE Transaction on Parallel and Distributed Systems.* **10**(7) 673-693.

[5] Galati M. 2011. Decomposition Methods in Integer Linear Programming. *Ph.D. Dissertation.*

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

[7] Galati M., Ralphs T. 2009. DIP - Decomposition for integer programming. https://projects.coin-or.org/Dip

[8] Jünger. M. Branch-and-Cut Algorithms for Combinatorial Optimization and Their Implementation in ABACUS. *Computational Combinatorial Optimization.* **2241**. 157-222

[9] Gamrath G., Lübbecke M.E. Experiments with a Generic Dantzig-Wolfe Decomposition for Integer Programs. *Symposium on Experimental Algorithms* (SEA 2010), LNCS, 6049, pp. 239-252, 2010, Springer, Berlin.

[10] Borndörfer, R. and Ferreira, C.E. and Martin, A. Decomposing matrices into blocks. *SIAM Journal on Optimization.* **9**(1), 236-269.

[11] Ferris M.C., Horn J.D. 1998. Partitioning mathematical programs for parallel solution. *Mathematical Programming.* **80**, 35-62.

[12] Aykanat C. Pinar A. Çatalyürek, Ü. V. 2004. Permuting sparse rectangular matrices into block-diagonal form. *SIAM Journal on scientific computing.* **25**(6), 1860-1879.

[13] Çatalyürek, Ü. V. 1999. Hypergraph models for sparse matrix partitioning and reordering. *Ph.D. Dissertation*

[14] J. Forrest. `http://www.coin-or.org/Clp`

[15] J. Forrest. `https://projects.coin-or.org/Cbc`

[16] Karypis G. 1998. `http://glaros.dtc.umn.edu/gkhome/views/metis`

[17] Çatalyürek, Ü. V. 1999. `http://bmi.osu.edu/~umit/software.html.`

[18] Çatalyürek, Ü. V. 2011. PaToH manual. `http://bmi.osu.edu/~umit/PaToH/manual.pdf`

[19] Çatalyürek. `http://bmi.osu.edu/~umit/PaToH/results.html`