Konrad-Zuse-Zentrum für Informationstechnik Berlin
Takustraße 7, D-14195 Berlin-Dahlem, Germany

Ralf Borndörfer   Carlos E. Ferreira   Alexander Martin

Decomposing Matrices into Blocks

To appear in SIAM Journal on Optimization.

Preprint SC-97-15 (March 1997, Revised October 1997)

Decomposing Matrices into Blocks*

Ralf Borndörfer** Carlos E. Ferreira† Alexander Martin**

Abstract. In this paper we investigate whether matrices arising from linear or integer programming problems can be decomposed into so-called *bordered block diagonal form.* More precisely, given some matrix A, we try to assign as many rows as possible to some number β of blocks of size κ such that no two rows assigned to different blocks intersect in a common column. Bordered block diagonal form is desirable because it can guide and speed up the solution process for linear and integer programming problems. We show that various matrices from the LP- and MIP-libraries `Netlib` and `Miplib` can indeed be decomposed into this form by computing optimal decompositions or decompositions with proven quality. These computations are done with a branch-and-cut algorithm based on polyhedral investigations of the matrix decomposition problem. In practice, however, one would use heuristics to find a good decomposition. We present several heuristic ideas and test their performance. Finally, we investigate the usefulness of optimal matrix decompositions into bordered block diagonal form for integer programming by using such decompositions to guide the branching process in a branch-and-cut code for general mixed integer programs.

Keywords. block structure of a sparse matrix, matrix decomposition, integer programming, polyhedral combinatorics, cutting planes

Mathematics Subject Classification (MSC 1991). 90C10, 65F50

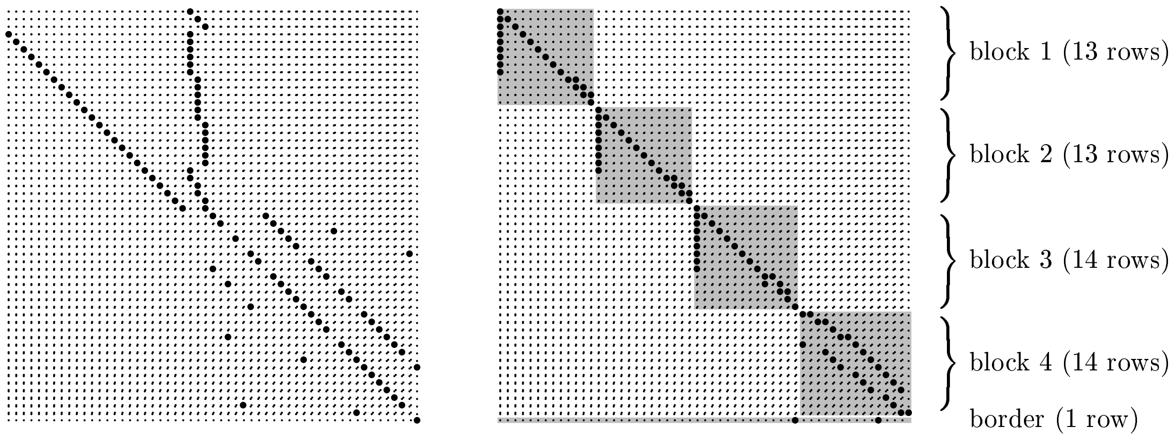

Figure 1: Decomposing a matrix into bordered block diagonal form.

1 Introduction

We consider in this paper the following *matrix decomposition problem.* Given some matrix A, some number β of *blocks* (sets of rows), and some *capacity* κ (maximum block-size); try to assign as many rows as possible to the blocks such that (i) each row is assigned to at most one block, (ii) each block contains at most κ rows, and (iii) no two rows in different blocks have a common nonzero entry in a column. The set of rows that are not assigned to any block is called the *border.*

An equivalent statement of the problem in matrix terminology is as follows: Try to decompose the matrix into *bordered block diagonal form* with β blocks of capacity at most κ. The decomposition is considered

*This work was supported by the Fundação Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) and the German Academic Exchange Service (DAAD), program PROBRAL.
**Konrad-Zuse-Zentrum Berlin, Takustraße 7, 14195 Berlin, Germany, `<surname>@zib.de`
†Univ. de São Paulo, Rua do Matão, 1010, 05508-970 — São Paulo, SP, Brazil, `cef@ime.usp.br`

the better, the smaller the number of rows in the border is; in the best case the border will be empty and the matrix decomposes into block diagonal form. Figure 1 shows on its left side the structure of a 55×55 non-symmetric matrix, namely, an optimal basis matrix of the `Netlib`-problem `recipe`. The right side of Figure 1 shows an optimal decomposition of this matrix into four blocks of capacity $\lceil 55/4 \rceil = 14$. To make the block structure of the decomposition visible, we have not only permuted the rows such that rows assigned to the same block appear consecutively, but also the columns. In this case, the blocks turn out to be almost square of sizes 13×13, 13×13, 14×14, and 14×15, but in general this does not need to be the case. The border consists of only one row that could not be assigned to any block.

The matrix decomposition problem fits into the general context of reordering matrices to *special forms*. Special forms are well studied in the literature because they can be exploited by solution methods for linear equation systems, for example by LU- or Cholesky factorization, or by conjugate gradient methods. The two main points of interest are that special forms allow (i) to control fill-in (bordered block diagonal form, in particular, restricts fill-in to the blocks and the border) and (ii) independent processing of individual blocks by parallel algorithms.

Methods to obtain special forms, including (bordered) block diagonal form, are widely discussed in the literature of computational linear algebra, see, for instance, Duff, Erisman, and Reid [1986], Kumar, Grama, Gupta, and Karypis [1994], or Gallivan, Heath, Ng, Ortega, Peyton, Plemmons, Romine, Sameh, and Voigt [1990]. The matrices studied in this context mainly arise from the discretization of partial differential equations. Some newer publications deal with matrices that appear in interior point algorithms for linear programs, see Gupta [1996] and Rothberg and Hendrickson [1996].

The applications we have in mind are different. We are interested in matrices arising from *(mixed) integer programs* (MIPs). Such matrices often have potential for decomposition into bordered block diagonal form for two reasons. First, such matrices are sparse and generally have a small number of nonzero entries in each column. Second, bordered block diagonal form comes up in a natural way in many real world MIPs. The problems are often composed of small blocks to model decisions in a division of a company, in a technical unit, or in a time period. These individual blocks are linked by a couple of constraints that model possible interactions to yield an appropriate model that covers the whole company, technical process, or time horizon. Examples of this type are production planning problems like the unit commitment problem in power generation, where the blocks correspond to single unit subproblems and the border is given by load balance and reserve constraints, see Sheble and Fahd [1994] for a literature synopsis and Dentcheva, Gollmer, Möller, Römisch, and Schultz [1997] for a recent application, multicommodity flow problems that come up, for example, in vehicle scheduling, see Löbel [1997], classes of combinatorial programs like the Steiner-tree packing problem, see Grötschel, Martin, and Weismantel [1996], or, recently, scenario decompositions of stochastic mixed integer programs, see Carøe and Schultz [1996].

Bordered block diagonal form helps to *accelerate* the solution process of integer programs in several ways. First, to solve the LP-relaxation of an integer program; here bordered block diagonal form can speed up the required linear algebra (both if a simplex type method or an interior point method is used). Second, it can be used to improve the polyhedral description of the set of feasible points of a MIP. For example, given a block decomposition and taking one constraint from each block plus an according number from the border results in the structure of a generalized assignment or multiple knapsack problem (see Gottlieb and Rao [1990] and Ferreira, Martin, and Weismantel [1996]) whose facets are valid inequalities for the MIP under consideration. Third, block decomposition can be used in a branch-and-bound or -cut algorithm to guide branching decisions: Decomposing the transposed constraint matrix A^T will identify the columns in the border as linking columns that are interesting candidates for branching.

In this paper we develop a branch-and-cut algorithm for solving the matrix decomposition problem. Of course the expected running time of such an algorithm will neither permit its usage within a parallel LU-factorization nor within a branch-and-cut algorithm for general MIPs. Our aim is rather to have a tool at hand that in principle obtains an *optimal* bordered block diagonal form. We can then evaluate whether this special matrix structure indeed helps in solving general integer programs, and we can evaluate the success of decomposition heuristics that try to obtain (bordered) block diagonal form and that could be used, for instance, within a parallel LU-factorization framework.

The paper is organized as follows. In Section 2 we formulate the matrix decomposition problem as a 0/1 linear program and discuss connections to related combinatorial optimization problems, namely, node separation problems in graphs, the set packing, and the set covering problem. Section 3 is devoted to a polyhedral investigation of the matrix decomposition problem and presents (new) valid and facet defining inequalities. In the branch-and-cut Section 4 we present our matrix decomposition algorithm including separation routines, primal heuristics, preprocessing, and other aspects of the implementation. We use this code in Section 5 to decompose optimal basis matrices of linear programs taken from the `Netlib` (available by anonymous `ftp` from `ftp://netlib2.cs.utk.edu`), to decompose matrices arising from mixed integer programs from the `Miplib` (available from `URL http://www.caam.rice.edu:80/~bixby/miplib/miplib.html`), and to solve some equipartition problems investigated by Nicoloso and Nobili [1992].

2 Integer Programming Formulation and Related Problems

Consider an instance (A, β, κ) of the matrix decomposition problem where $A \in \mathbb{R}^{m \times n}$ is some real matrix, $\beta \in \mathbb{N}$ is the number of blocks and $\kappa \in \mathbb{N}$ is the block capacity. We introduce for each row $i = 1, \ldots, m$ and block $b = 1, \ldots, \beta$ a binary variable x_i^b that has value 1 if row i is assigned to block b and 0 otherwise. Then the matrix decomposition problem (A, β, κ) can be formulated as the 0/1 linear program (IP) that is stated on this page.

Inequalities (i) guarantee that each row is assigned to at most one block. Constraints (ii) ensure that the number of rows assigned to a particular block b does not exceed its capacity. Finally, (iii) expresses that two rows i and j must not be assigned to different blocks if both have a nonzero entry in some common column. These three sets of inequalities plus the bounds (iv) and the integrality constraints (v) establish a one-to-one correspondence between feasible solutions of (IP) and block decompositions of the matrix A into β blocks of capacity κ. In the sequel we will also call a vector $x \in \mathbb{R}^{m \times \beta}$ a *block decomposition* if it is feasible for (IP). Note that formulation (IP) as it stands is not polynomial, since the number of variables $m\beta$ is not polynomial in the encoding length of β. However, we may assume without loss of generality $\beta \leq m$, because no more than m rows will be assigned. We also assume that the block capacity is at least one ($\kappa \geq 1$) and that we have at least two blocks ($\beta \geq 2$).

$$
\text{(IP)} \qquad
\begin{array}{lll}
\max & \displaystyle\sum_{i=1}^{m}\sum_{b=1}^{\beta} x_i^b & \\
\text{(i)} & \displaystyle\sum_{b=1}^{\beta} x_i^b \leq 1, & \text{for } i = 1, \ldots, m; \\
\text{(ii)} & \displaystyle\sum_{i=1}^{m} x_i^b \leq \kappa, & \text{for } b = 1, \ldots, \beta; \\
\text{(iii)} & x_i^b + x_j^{b'} \leq 1, & \text{for } b, b' = 1, \ldots, \beta,\ b \neq b' \text{ and} \\
 & & \text{for } i, j = 1, \ldots, m,\ i \neq j \text{ such that} \\
 & & a_{ik} \neq 0 \neq a_{jk} \text{ for some } k \in \{1, \ldots, n\}; \\
\text{(iv)} & 0 \leq x_i^b \leq 1, & \text{for } i = 1, \ldots, m,\ b = 1, \ldots, \beta; \\
\text{(v)} & x_i^b \text{ integer}, & \text{for } i = 1, \ldots, m,\ b = 1, \ldots, \beta.
\end{array}
$$

A first observation about (IP) is that different matrices A can give rise to the same integer program or, in other words, different matrices can be decomposed in exactly the same way. In fact, such matrices form equivalence classes as can be seen by considering the *(column) intersection graph* $G(A)$ of an $m \times n$-matrix A as introduced by Padberg [1973]. $G(A)$ has the set $\{1, \ldots, n\}$ of column indices of A as its node set and there is an edge ij between two columns i and j if they have a common nonzero entry in some row. Applying this concept to the transposed matrix A^T, we obtain the *row intersection graph* $G(A^T)$ of A where two rows i and j are joined by an edge ij if and only if they have nonzero entries in a common column. But then the edges of $G(A^T)$ give rise to the inequalities (IP) (iii) and we have that for fixed β and κ two matrices A and A' have the same row intersection graph if and only if the corresponding integer programs (IP) are equal.

The matrix decomposition problem is related to several other combinatorial optimization problems. First, the problem can be interpreted in terms of the row intersection graph as a *node separator problem*. To see this, let $G(A^T) = (V, E)$ and consider some block decomposition x. The set $S := \{i \in V : \sum_{b=1}^{\beta} x_i^b = 0\}$ of rows in the border is a node separator in $G(A^T)$ such that the graph obtained by deleting all nodes in S and all its adjacent edges decomposes into at most β parts, each of cardinality at most κ. Conversely, each node separator in $G(A^T)$ with these properties gives rise to a block decomposition for (A, β, κ). Various node separator problems have been studied in the literature. Lengauer [1990] gives a survey and discusses applications in VLSI design, Duff, Erisman, and Reid [1986] and Gallivan, Heath, Ng, Ortega, Peyton, Plemmons, Romine, Sameh, and Voigt [1990] emphasize heuristic methods for use in computational linear algebra. Lower bounds on the size of a node separator in a general graph are rather rare. The only results we are aware of are due to Pothen, Simon, and Liou [1990] and Helmberg, Mohar, Poljak, and Rendl [1995], who use Eigenvalue methods to derive non-trivial lower bounds on the size of a node separator for $\beta = 2$, if lower bounds on the size of the blocks are imposed.

A second connection exists to *set packing*, and this relationship is two-fold. On the one hand, matrix decomposition is a generalization of set packing, because feasible solutions (stable sets) of some set packing problem $\max\{\mathbb{1}^T x : Ax \leq \mathbb{1}, x \in \{0,1\}^n\}$, $A \in \{0,1\}^{m\times n}$, correspond to solutions of the matrix decomposition problem $(A^T, m, 1)$ of the same objective value and vice versa. This shows that the matrix decomposition problem is $\mathcal{NP}$-hard. On the other hand, we obtain a *set packing relaxation* of the matrix decomposition problem by deleting the block capacity constraints (ii) from the formulation (IP). All inequalities that are valid for this relaxation are also valid for the matrix decomposition problem and we will use some of them (namely clique- and cycle-inequalities) as cutting planes in our branch-and-cut algorithm. Note, however, that the set packing relaxation allows assignment of all rows to any single block and our computational experiments seem to indicate that these cuts are rather weak.

A close connection exists also to *set covering* via complementing variables. To see this we rewrite (IP), substituting each capacity constraint (ii) by $\binom{m}{\kappa+1}$ inequalities that sum over all subsets of cardinality $\kappa+1$ of variables $\{x_1^b, \ldots, x_m^b\}$ for some block b and each constraint in (i) by $\binom{\kappa}{2}$ inequalities that sum over all pairs of variables in $\{x_i^1, \ldots, x_i^\beta\}$. Replacing all variables x_i^b by $1-y_i^b$, one obtains the following set covering problem:

$$
(\mathrm{IP}_c)\qquad
\begin{array}{lll}
\min & \displaystyle\sum_{i=1}^{m}\sum_{b=1}^{\beta} y_i^b & \\
\text{(i)} & y_i^b + y_i^{b'} \geq 1, & \text{for } b, b' = 1, \ldots, \beta,\ b \neq b' \text{ and} \\
 & & \text{for } i = 1, \ldots, m; \\
\text{(ii)} & \displaystyle\sum_{i \in I} y_i^b \geq 1, & \text{for } b = 1, \ldots, \beta \text{ and} \\
 & & \text{for } I \subseteq \{1, \ldots, m\} \text{ with } |I| = \kappa + 1; \\
\text{(iii)} & y_i^b + y_j^{b'} \geq 1, & \text{for } b, b' = 1, \ldots, \beta,\ b \neq b' \text{ and} \\
 & & \text{for } i, j = 1, \ldots, m,\ i \neq j \text{ such that} \\
 & & a_{ik} \neq 0 \neq a_{jk} \text{ for some } k \in \{1, \ldots, n\}; \\
\text{(iv)} & 0 \leq y_i^b \leq 1, & \text{for } i = 1, \ldots, m,\ b = 1, \ldots, \beta; \\
\text{(v)} & y_i^b \text{ integer}, & \text{for } i = 1, \ldots, m,\ b = 1, \ldots, \beta.
\end{array}
$$

This shows that the matrix decomposition problem is a (special) set covering problem. For the case of two blocks, this formulation has been used by Nicoloso and Nobili [1992] for the solution of the *matrix equipartition problem*. The matrix equipartition problem is the matrix decomposition problem for $\beta = 2$ and $\kappa = \lfloor m/2 \rfloor$, plus the additional equipartition constraint

$$
\sum_{i=1}^{m} x_i^1 = \sum_{i=1}^{m} x_i^2, \quad \text{or, in complemented variables,} \quad \sum_{i=1}^{m} y_i^1 = \sum_{i=1}^{m} y_i^2,
$$

that states that the two blocks of the decomposition must have equal size.

3 Polyhedral Investigations

Associated to the IP-formulation (IP) of the matrix decomposition problem is the polytope

$$
(1)\qquad P(A, \beta, \kappa) := \operatorname{conv}\{x \in \mathbb{R}^{m\times\beta} : x \text{ satisfies (IP) (i) to (v)}\},
$$

given by the convex hull of all block decompositions. We study in this section the structure of $P(A, \beta, \kappa)$ to derive classes of valid and facet defining inequalities for later use as cutting planes. We start by determining its dimension.

3.1 Proposition (Dimension) *$P(A, \beta, \kappa)$ is full dimensional.*

Proof. The vector 0 and all unit vectors $e_i^b \in \mathbb{R}^{m \times \beta}$ are feasible, i.e., are in $P(A, \beta, \kappa)$, and affinely independent. $\square$

This means that the facets of $P(A, \beta, \kappa)$ are uniquely determined up to a scalar factor (see Schrijver [1986]). Two further easy observations are gathered in the following remark.

3.2 Remark *(i) The non-negativity inequalities $x_i^b \geq 0$ are facet defining for all $i = 1, \dots, m$ and all $b = 1, \dots, \beta$.*

(ii) All facet defining inequalities $a^T x \leq \alpha$ that are not non-negativity constraints satisfy $a \geq 0$ and $\alpha > 0$.

Remark 3.2 (i) is proven in the same way as Theorem 3.1 and Remark 3.2 (ii) is a consequence of the down monotonicity of $P(A, \beta, \kappa)$.

Facet-defining inequalities have another interesting property. Consider some vector $x \in \mathbb{R}^{m \times \beta}$, some *permutation σ of the blocks* $\{1, \dots, \beta\}$, and define the vector $\bar{x} \in \mathbb{R}^{m \times \beta}$ by

$$\bar{x}_i^b := x_i^{\sigma(b)},$$

for $i = 1, \dots, m, b = 1, \dots, \beta$. We will use in the sequel the symbol $\sigma(x)$ to denote the vector $\bar{x}$ that arises from x by applying the block permutation σ. Then $\sigma(x) = \bar{x}$ is a feasible block decomposition if and only if x is. This simple observation has two consequences. First, it implies that $a^T x \leq b$ is a facet of $P(A, \beta, \kappa)$ if and only if its block-wise permutation $\sigma(a)^T x \leq b$ is. Facets arising from each other via block permutations can thus be viewed as forming a single class that can be represented by a single member. Or, to put it in a more negative way, each facet can and will be "blown up" by block permutations to a whole set of combinatorially essentially identical conditions. Second, the objective function of the matrix decomposition problem is invariant under block permutation and thus the matrix decomposition problem is dual degenerate (has multiple optima). Both dual degeneracy and the large number of permutable facets cause difficulties in our branch-and-cut algorithm and we will have to control the number of cuts generated and to handle stalling of the objective value.

The next two subsections list the results of our polyhedral investigations in the form of valid and facet defining inequalities. We distinguish between inequalities $a^T x \leq b$ that are *invariant under block permutations* or, equivalently, have the same coefficients $a_i^b = a_i^{b'}$ for all blocks $b \neq b'$ and row indices i, and *block-discernible inequalities* that do not have this property and distinguish different blocks. It will turn out that most of the block-discernible inequalities will be inherited from the stable set relaxation of the matrix decomposition problem, while the block-invariant constraints are related to an "aggregated" version of the problem. In both subsections we want to assume $\kappa \geq 2$, because otherwise the matrix decomposition problem is a (special) set packing problem.

3.1 Block-Discernible Inequalities

We saw in Section 2 that we obtain a set packing relaxation of the matrix decomposition problem by dropping the block capacity constraints (ii) from the integer program (IP). The column intersection graph associated to the matrix $\mathrm{IP}_{(\mathrm{i}),(\mathrm{iii})}$ formed by the left-hand sides of the constraints (IP) (i) and (iii) has the set of possible row assignments $\{1, \dots, m\} \times \{1, \dots, \beta\}$ as its node set. A (conflict) edge exists between two assignments (i, b) and (j, b'), if rows i and j cannot be simultaneously assigned to the blocks b and b', i.e., either if $i = j$ and $b \neq b'$ or if $i \neq j$, $b \neq b'$, and rows i and j have a common nonzero entry in some column of A. We want to call this graph the *conflict graph* associated to the matrix decomposition problem (A, β, κ) and denote it by $G_c(A, \beta)$. In formulas: $G_c(A, \beta) = G(\mathrm{IP}_{(\mathrm{i}),(\mathrm{iii})})$. This graph allows us to interpret the inequality classes (i) and (iii) of (IP) as *clique inequalities* of the set packing relaxation corresponding to the matrix decomposition problem as also introduced by Padberg [1973].

3.3 Theorem (Clique) *Let $G_c(A, \beta) = (V, E)$ and $Q \subseteq V$. The inequality*

$$\sum_{(i,b) \in Q} x_i^b \leq 1$$

is valid for $P(A, \beta, \kappa)$ if and only if Q is a clique in $G_c(A, \beta)$. It is facet defining if and only if Q is a maximal clique in $G_c(A, \beta)$.

Proof. The validity part is obvious. It remains to show that it is facet defining if and only if Q is a maximal clique.

Suppose first that Q is not maximal but contained in a larger clique Q'. But then $\sum_{(i,b)\in Q} x_i^b \le 1$ is the sum of the inequality $\sum_{(i,b)\in Q'} x_i^b \le 1$ and the non-negativity constraints $x_i^b \ge 0$ for $(i,b) \in Q' \setminus Q$ and cannot be facet defining.

Assume now that Q is maximal. We will construct a set of $m\beta$ affinely independent block decompositions for which the inequality is tight. $|Q|$ such affinely independent vectors are the unit vectors e_i^b with $(i,b) \in Q$. For each other assignment $(j,b') \notin Q$ there exists some assignment (i,b) in Q that is not in conflict with (j,b'), since Q is a maximal clique. Thus, the vector $e_j^{b'} + e_i^b$ is the incidence vector of a feasible block decomposition for which the inequality is tight. (Note that we assumed $\kappa \ge 2$ at the beginning of this section for the case $b = b'$.) The resulting $m\beta - |Q|$ characteristic vectors obtained in this way plus the $|Q|$ vectors constructed in the beginning are affinely independent. □

In the spirit of Theorem 3.3, (IP) (i) and (iii) are both clique inequalities and do not represent two different types of inequalities. The separation problem for clique inequalities is a maximum-weight clique problem and thus $\mathcal{NP}$-hard, see Garey and Johnson [1979]. But some subclasses can be separated efficiently. One such class that we use in our implementation are the *two-partition inequalities*

$$\sum_{b\in B} x_i^b + \sum_{b'\notin B} x_j^{b'} \le 1,$$

that are defined for all sets of blocks $B \subseteq \{1,\dots,\beta\}$ and all pairs of non-disjoint rows i, j. Polynomial separation of this class is by inspection: Given i and j, we examine for each block b the variables x_i^b and x_j^b. If $x_i^b > x_j^b$, we add b to the set B, otherwise to its complement. Note that for the case of two blocks ($\beta = 2$), the two-partition inequalities are exactly the inequalities (IP) (i) and (iii) and, moreover, these are already all clique inequalities. In particular, separation of clique inequalities is polynomial for $\beta = 2$. In general, maximal cliques in $G_c(A,\beta)$ are of the form $\{(i_1,b_1),\dots,(i_\beta,b_\beta)\}$, where the blocks $b_k, k = 1,\dots,\beta$ are mutually different and the set of rows $\{i_1,\dots,i_\beta\}$ forms a clique in $G(A^T)$. Thus all maximal cliques in $G_c(A,\beta)$ are of size β.

Another class inherited from the set packing relaxation are the *cycle inequalities*.

3.4 Theorem (Odd Cycle) *If C is an odd cycle in $G_c(A,\beta)$, then the cycle inequality*

$$\sum_{(i,b)\in C} x_i^b \le \lfloor |C|/2 \rfloor$$

is valid for $P(A,\beta,\kappa)$.

Analogously to the set packing case, see again Padberg [1973], the odd cycle inequality is facet defining for its support if C is an odd hole (has no chords) and $|C|/2 \le \kappa$. These conditions are, however, not necessary. Cycle inequalities can be separated in polynomial time using the algorithm of Lemma 9.1.11 in Grötschel, Lovász, and Schrijver [1988].

Along the same lines as for the clique and cycle inequalities, the matrix decomposition polytope clearly also inherits all other packing inequalities. But not only set packing, also set covering inequalities for (IP_c) can be applied (note that complementing variables preserves validity and dimension of the induced face), see Nobili and Sassano [1989]. We do, however, not use any of them for our computations.

We close this section investigating the *block capacity constraints* (IP) (ii) which are not inherited from the set packing polytope or the set covering polytope.

3.5 Theorem (Block Capacity) *The block capacity constraint*

$$\sum_{i=1}^{m} x_i^b \le \kappa$$

is facet defining for $P(A,\beta,\kappa)$ if and only if $|\gamma(i)| \le m - \kappa$ holds for every row i (where $\gamma(i)$ denotes all nodes adjacent to i in $G(A^T)$).

Proof. We first show that the inequality is facet defining if the above mentioned condition holds. To this purpose, let $a^T x \leq \alpha$ be a valid inequality that induces a facet such that $\{x \in P(A,\beta,\kappa) | \sum_{i=1}^m x_i^b = \kappa\} \subseteq \{x \in P(A,\beta,\kappa) | a^T x = \alpha\}$. We will show that the two inequalities are the same up to a positive scalar multiplicative factor.

Define x by

$$x_i^{b'} = \begin{cases} 1, & \text{if } 1 \leq i \leq \kappa,\ b' = b; \\ 0, & \text{else.} \end{cases}$$

x is a feasible block decomposition that assigns the first κ rows to block b. x satisfies the block capacity constraint with equality and thus $a^T x = \alpha$. Now observe that, for all $1 \leq i \leq \kappa < j \leq m$, the vector $x - e_i^b + e_j^b$ is also a feasible assignment that is tight for the block capacity inequality. It follows that $a_i^b = a_j^b$ for all $1 \leq i, j \leq m$.

Now consider assigning some row j to a block $b' \neq b$. By the assumption $|\gamma(j)| \leq m - \kappa$, there is a set $R(j)$ of κ rows not adjacent to j. But then $\sum_{i \in R(j)} e_i^b$ and $\sum_{i \in R(j)} e_i^b + e_j^{b'}$ are both feasible decompositions that satisfy the block capacity constraint with equality and thus $a_j^{b'} = 0$, completing the first part of the proof.

It remains to prove the converse direction. If there is some row j with $|\gamma(j)| > m - \kappa$, the inequality $\sum_{i=1}^m x_i^b + \sum_{b' \neq b} x_j^{b'} \leq \kappa$ is valid. But then the block capacity constraint can be obtained by summing up this inequality with $\sum_{b' \neq b} x_j^{b'} \geq 0$, and therefore it cannot be facet defining. □

3.2 Block-Invariant Inequalities

We investigate in this section inequalities for the matrix decomposition polytope that are invariant under block permutation. Consider for each block decomposition x the "aggregated" vector

$$z(x) := \left(\sum_{b=1}^{\beta} x_1^b, \ldots, \sum_{b=1}^{\beta} x_m^b \right) \in \mathbb{R}^m .$$

$z(x)$ only records whether the matrix rows are assigned to some block or not, but no longer to which block. From a polyhedral point of view, the aggregated block decompositions give rise to an "aggregated" version of the block decomposition polytope

$$P_z(A,\beta,\kappa) := \operatorname{conv} \{z \in \mathbb{R}^m : \text{ there is } x \in P(A,\beta,\kappa) \text{ with } z = z(x)\}.$$

The aggregated polytope is interesting because any valid inequality $\sum_{i=1}^m a_i z_i \leq \alpha$ for $P_z(A,\beta,\kappa)$ can be "expanded" into an inequality $\sum_{i=1}^m a_i \sum_{b=1}^{\beta} x_i^b \leq \alpha$ that is valid for $P(A,\beta,\kappa)$. All inequalities in this subsection are of this type. Obviously, the expansion process yields inequalities that are invariant under block permutations, hence the name.

From a computational point of view, block-invariant inequalities are promising cutting planes, because the objective of the matrix decomposition problem can be written in terms of aggregated z-variables as $\mathbb{1}^T x = \mathbb{1}^T z(x)$. Thus, a complete description of $P_z(A,\beta,\kappa)$ would already allow us to determine the correct objective function value of the matrix decomposition problem and z-cuts will help to raise the lower bound of an LP-relaxation.

The aggregated polytope $P_z(A,\beta,\kappa)$ provides a model of the matrix decomposition problem that rules out degeneracy due to block permutations. While this is a very desirable property of the aggregated z-formulation, its drawback is that it is already $\mathcal{NP}$-complete to decide whether a given vector $z \in \{0,1\}^m$ is an aggregated block decomposition or not. (It can be shown that this is a bin-packing problem.) Our choice to use z-cuts within the x-model tries to circumvent this difficulty and combines the strengths of both formulations. We remark that degeneracy problems of this type arise also in block-indexed formulations of grouping problems in cellular manufacturing, where the difficulty can be resolved by means of alternative formulations, see Crama and Oosten [1996].

We already know one example of an expanded aggregated constraint: Expanding the inequality $z_i \leq 1$ for the aggregated block decomposition polytope yields the block assignment constraint (IP) (i) $\sum_{b=1}^{\beta} x_i^b \leq 1$ that we have analyzed in the previous subsection. More inequalities are derived from the observation that adjacent rows (with respect to $G(A^T)$) can only be assigned to the same block. A first example of this sort of inequalities are the *z-cover inequalities*.

3.6 Theorem (z-Cover) *Let $G(A^T) = (V, E)$ and let $W \subseteq V$ be a set of rows of cardinality $\kappa + 1$. Then, the z-cover inequality*

$$\sum_{i \in W} \sum_{b=1}^{\beta} x_i^b \le \kappa$$

is valid for $P(A, \beta, \kappa)$ if and only if $(W, E(W))$ is connected (where $E(W)$ denotes all edges with both endpoints in W). It is facet defining for $P(A, \beta, \kappa)$ if and only if for each row $i \notin W$ the graph $(W \cup \{i\}, E(W \cup \{i\}))$ has an articulation point different from i.

Proof. The validity part is easy. Since $|W| = \kappa + 1$, not all rows can be assigned to the same block. If some rows of W are assigned to different blocks, there must be at least one row in W that is not assigned because $(W, E(W))$ is connected. Conversely, if W is not connected one easily finds a partition of W into two subsets that can be assigned to different blocks.

The proof that this inequality is facet defining if and only if for each row $i \notin W$ the graph $(W \cup \{i\}, E(W \cup \{i\}))$ has an articulation point different from i is analogous to the proof of Theorem 3.5. The condition guarantees that if row i is assigned to some block, the assignment can be extended in such a way that κ rows from W can be assigned to at least two blocks. On the other hand, if the condition is not satisfied for some $j \notin W$, the inequality $\sum_{i \in W \cup \{j\}} \sum_{b=1}^{\beta} x_i^b \le \kappa$ is valid, and thus the z-cover inequality cannot be facet defining. □

In the set covering model, z-cover inequalities correspond to constraints of the form $\sum_{i \in W} \sum_{b=1}^{\beta} y_i^b \ge 1$ that have been used by Nicoloso and Nobili [1992] for their computations. The separation problem is to find a tree of size $\kappa + 1$ of maximum node weight. This problem has been studied by Ehrgott [1992] and was shown to be $\mathcal{NP}$-hard using a reduction to the node-weighted Steiner-tree problem.

The z-cover inequalities are induced by trees, but it is possible to generalize them for subgraphs of higher connectivity.

3.7 Theorem (Generalized z-Cover) *Let $G(A^T) = (V, E)$ and let $W \subseteq V$ be a set of rows of cardinality $\kappa + k$ with $k \ge 1$. Then, the (generalized) z-cover inequality*

$$\sum_{i \in W} \sum_{b=1}^{\beta} x_i^b \le \kappa$$

is valid for $P(A, \beta, \kappa)$ if and only if $(W, E(W))$ is k-node connected. It is facet defining for $P(A, \beta, \kappa)$ if and only if for each row $i \notin W$ there exists some node cut N in $(W \cup \{i\}, E(W \cup \{i\}))$ of cardinality k with $i \notin N$.

The proof of this generalization follows exactly the lines of the proof of Theorem 3.6. In our branch-and-cut algorithm we restrict attention to the cases $k = 1$ and $k = 2$.

Closely related to the z-cover inequality is the *z-clique inequality*. Here, we consider some node set W that is not only k-node connected for some fixed k, but induces a complete subgraph. In this case the condition for being facet defining slightly changes.

3.8 Theorem (z-Clique) *If Q is a clique in $G(A^T)$, then the z-clique inequality*

$$\sum_{i \in Q} \sum_{b=1}^{\beta} x_i^b \le \kappa$$

is valid for $P(A, \beta, \kappa)$. It is facet-defining if and only if $|Q| \ge \kappa + 1$ and for each row $i \notin Q$ there exists a set of rows $R(i) \subseteq Q$, $|R(i)| = \kappa$, such that i is not adjacent in $G(A^T)$ to any node in $R(i)$.

Proof. The inequality is clearly valid. To show that it is facet defining given the mentioned conditions, let $a^T x \le \alpha$ define a facet such that

$$\{x \in P(A, \beta, \kappa) | \sum_{i \in Q} \sum_{b=1}^{\beta} x_i^b = \kappa\} \subseteq \{x \in P(A, \beta, \kappa) | a^T x = \alpha\}.$$

We will show that the two inequalities are the same up to a positive scalar multiplicative factor. To this purpose, consider any κ rows of Q. The block decomposition obtained by assigning these rows to some block b is feasible and tight for the z-clique inequality. Since $|Q| \geq \kappa+1$, we can use these solutions to show that $a_i^b = a_j^{b'}$ for all $i, j \in Q$ and for all blocks $b, b' \in \{1, \ldots, \beta\}$. Assuming that for each row $i \notin Q$ there exists a set of nodes $R(i) \subseteq Q$, $|R(i)| = \kappa$, that are not adjacent to i, we observe that for all $b' \neq b$, the vectors $\sum_{j \in R(i)} e_j^b$ and $\sum_{j \in R(i)} e_j^b + e_i^{b'}$ are valid block decompositions that satisfy the z-clique inequality with equality. It follows that $a_i^{b'} = 0$ for all $i \notin Q$, for all $b' \neq b$, and even for all blocks b', since b was arbitrary. This completes the first part of the proof.

If, on the other hand, Q has size less than or equal to κ, we obtain from (IP) (i) that the left hand side of the inequality is at most $|Q|$. Thus, the inequality is redundant and cannot define a facet. Suppose now the second condition is not satisfied, i. e., there is some $j \notin Q$ such that j is incident to at least $|Q| - \kappa + 1$ nodes in Q. This implies that $Q \cup \{j\}$ is at least $(|Q| - \kappa + 1)$-node connected. Theorem 3.7 states that $\sum_{i \in Q \cup \{j\}} \sum_{b=1}^{\beta} x_i^b \leq \kappa$ is valid and this implies that the z-clique inequality is redundant. □

The z-clique separation problem is again a max-clique problem and thus $\mathcal{NP}$-hard. In our implementation we check easily detectable special cases like the following so-called *big-edge inequalities*

$$\sum_{i \in \operatorname{supp}(A_{\cdot j})} \sum_{b=1}^{\beta} x_i^b \leq \kappa,$$

for all blocks b, where $A_{\cdot j}$ denotes the j-th column of A and $\operatorname{supp}(A_{\cdot j})$ its nonzero row indices. These inequalities can be separated by inspection.

Another way to generalize the z-cover inequalities is by looking at node induced subgraphs that consist of several components. This idea, that gives rise to the class of *bin-packing inequalities*, came up in our computational experiments. Starting point is again a set of rows W that induces a subgraph of $G(A^T) = (V, E)$. Suppose $(W, E(W))$ consists of l connected components of sizes (in terms of nodes) $a_1, \ldots, a_l$. We can then associate a *bin-packing problem* with $(W, E(W))$, β, and κ in the following way: There are l items of sizes $a_1, \ldots, a_l$, and β bins of capacity κ each. The problem is to put all the items into the bins such that no bin holds items of a total size that exceeds the capacity κ. If this is not possible, we can derive a valid inequality for $P(A, \beta, \kappa)$.

3.9 Theorem *Let $G(A^T) = (V, E)$ and $W \subseteq V$ be some subset of rows. If the bin packing problem associated to $(W, E(W))$, β, and κ has no solution, the bin-packing inequality*

$$\sum_{i \in W} \sum_{b=1}^{\beta} x_i^b \leq |W| - 1$$

is valid for $P(A, \beta, \kappa)$.

Proof. Consider some block decomposition x. If at least one row in W is not assigned to some block, the inequality is obviously satisfied. Otherwise all rows that belong to the same (connected) component of $(W, E(W))$ must be assigned to the same block. This yields a solution to the bin packing problem associated to $(W, E(W))$, β, and κ, a contradiction. □

We do not know any reasonable conditions that characterize when the bin packing inequalities are facet defining. Bin-packing separation is $\mathcal{NP}$-hard, see Garey and Johnson [1979].

Next we give another class of *z-cycle inequalities* that generalize the cycle inequalities of the set packing polytope.

3.10 Theorem (z-Cycle) *Let $G(A^T) = (V, E)$ and $C \subseteq V$ be a cycle in $G(A^T)$ of cardinality at least $\kappa + 1$. Then the z-cycle inequality*

$$\sum_{i \in C} \sum_{b=1}^{\beta} x_i^b \leq |C| - \left\lceil \frac{|C|}{\kappa + 1} \right\rceil$$

is valid for $P(A, \beta, \kappa)$.

The z-cycle inequality is valid because at least every $(\kappa+1)$-st node cannot be assigned to a block. One can also show that the inequality is facet defining for its support under certain rather restrictive conditions, for example, if C is an odd hole, $|C| \neq 0 \mod (\kappa+1)$, and the right-hand side is less than $\beta\kappa$. z-Cycle separation can be reduced to the TSP and is thus $\mathcal{NP}$-hard.

Our next class of inequalities comes up in several instances in our test set.

3.11 Theorem (Composition of Cliques (COQ)) *Let $G(A^T) = (V, E)$ and consider p mutually disjoint cliques $Q_1, \ldots, Q_p \subseteq V$ of size q and q mutually disjoint cliques $P_1, \ldots, P_q \subseteq V$ of size p such that $|P_i \cap Q_j| = 1$ for all i, j. Let $W = \cup_{i=1}^{p} Q_i$. Then, the following inequality is valid for $P(A, \beta, \kappa)$:*

$$\sum_{i \in W} \sum_{b=1}^{\beta} x_i^b \leq \max_{\substack{\{r \in \mathbb{N}^\beta, s \in \mathbb{N}^\beta: \\ \sum r_b = p, \sum s_b = q\}}} \sum_{b=1}^{\beta} \min\{\kappa, r_b s_b\} =: \alpha(p, q, \beta, \kappa). \tag{2}$$

Proof. Consider a block decomposition x and let

$$\begin{aligned} \overline{r}_b &:= |\{j : \textstyle\sum_{i \in Q_j} x_i^b \geq 1, j \in \{1, \ldots, p\}\}|, \\ \overline{s}_b &:= |\{j : \textstyle\sum_{i \in P_j} x_i^b \geq 1, j \in \{1, \ldots, q\}\}|, \end{aligned}$$

for $b = 1, \ldots, \kappa$. Because Q_j and P_j are all cliques, we have that $\sum_{b=1}^{\beta} \overline{r}_b \leq p$ and $\sum_{b=1}^{\beta} \overline{s}_b \leq q$. Since $|P_i \cap Q_j| = 1$ for all i, j it follows that $\sum_{i \in W} x_i^b \leq \overline{r}_b \overline{s}_b$. Thus,

$$\begin{aligned} \sum_{i \in W} \sum_{b=1}^{\beta} x_i^b &= \sum_{b=1}^{\beta} \sum_{i \in W} x_i^b \\ &\leq \sum_{b=1}^{\beta} \min\{\kappa, \overline{r}_b \overline{s}_b\} \\ &\leq \max_{\substack{\{r \in \mathbb{N}^\beta, s \in \mathbb{N}^\beta: \\ \sum r_b \leq p, \sum s_b \leq q\}}} \sum_{b=1}^{\beta} \min\{\kappa, r_b s_b\} \\ &= \max_{\substack{\{r \in \mathbb{N}^\beta, s \in \mathbb{N}^\beta: \\ \sum r_b = p, \sum s_b = q\}}} \sum_{b=1}^{\beta} \min\{\kappa, r_b s_b\}, \end{aligned}$$

showing the statement. □

The right hand side of (2) is quite complicated, and we do not even know whether it can be computed in polynomial time. For $\beta = 2$ the right hand side looks more tractable:

$$\sum_{i \in W} \sum_{b=1}^{\beta} x_i^b \leq \max_{\substack{r=0,\ldots,p \\ s=0,\ldots,q}} \left(\min\{\kappa, rs\} + \min\{\kappa, (p-r)(q-s)\}\right). \tag{3}$$

But we do not know a closed formula in this case either. An interesting special case is $p = 2$. Here the graph $(W, E(W))$ consists of two disjoint cliques that are joint by a perfect matching. Suppose further $q < \kappa < 2q$. Then the right hand side of (3) reads

$$\begin{aligned} &\max\{0, \max_{s=0,\ldots,q} (\min\{\kappa, s\} + \min\{\kappa, q-s\}), \min\{\kappa, 2q\}\} \\ &= \max\{0, q, \kappa\} = \kappa. \end{aligned}$$

In this case (2) turns out to be even facet defining if we require in addition that each node $i \notin W$ has at most $2q - \kappa$ neighbors in W, i, e., $|\gamma(i) \cap W| \leq 2q - \kappa$.

The development of our heuristic separation routine for COQ inequalities resulted in a slight generalization of this class. The support graphs of the left-hand sides of these *extended composition of clique inequalities* are COQs where some nodes have been deleted, the right-hand sides are left unchanged.

3.12 Theorem (Extended Composition of Cliques (xCOQ)) *Let $G(A^T) = (V, E)$ and consider p mutually disjoint non-empty cliques $Q_1, \ldots, Q_p \subseteq V$ of size at most q and q mutually disjoint non-empty cliques $P_1, \ldots, P_q \subseteq V$ of size at most p such that*

(i) $|P_i \cap Q_j| \leq 1$ *for all* i, j *and*

(ii) $\sum_{i=1}^{q} \sum_{j=1}^{p} |P_i \cap Q_j| = \sum_{i=1}^{q} |P_i| = \sum_{j=1}^{p} |Q_j|,$

i.e., every element in one of the sets P_i appears in exactly one of the sets Q_j and vice versa. Let $W = \cup_{i=1}^{p} Q_i$. Then, the following inequality is valid for $P(A, \beta, \kappa)$:

$$\sum_{i \in W} \sum_{b=1}^{\beta} x_i^b \leq \alpha(p, q, \beta, \kappa). \tag{4}$$

Proof. The proof works by turning $P_1, \ldots, P_q$ and $Q_1, \ldots, Q_p$ into a proper COQ by adding some nodes that correspond to "artificial rows" and projecting the resulting inequality down to the original space of variables.

Let

$$\delta := \sum_{i=1}^{q} \sum_{j=1}^{p} (1 - |P_i \cap Q_j|)$$

be the number of nodes that "miss" to turn $P_1, \ldots, P_q$ and $Q_1, \ldots, Q_p$ into a COQ and add a row $\mathbb{1}^T$ of all ones to A for each of them to obtain a matrix $\bar{A}$ such that

$$\bar{A}_{i\cdot} = A_{i\cdot}, \quad i = 1, \ldots, m \qquad \text{and} \qquad \bar{A}_{i\cdot} = \mathbb{1}^T, \quad i = m+1, \ldots, m+\delta.$$

Consider the matrix decomposition problem $(\bar{A}, \beta, \kappa)$. Its row intersection graph $G(\bar{A}^T)$ contains $G(A^T)$ as a subgraph and the additional artificial nodes in $G(\bar{A}^T)$ are incident to every node of $G(\bar{A}^T)$ that corresponds to a row that is not all zero (except itself).

The sets $P_1, \ldots, P_q$ and $Q_1, \ldots, Q_p$ are again cliques in $G(\bar{A}^T)$. Associating each of the artificial nodes $i = m+1, \ldots, m+\delta$ to a different index pair ij such that $|P_i \cap Q_j| = 0$ and adding this node to both P_i and Q_j, we can extend $P_1, \ldots, P_q$ and $Q_1, \ldots, Q_p$ to a COQ $\overline{P}_1, \ldots, \overline{P}_q$ and $\overline{Q}_1, \ldots, \overline{Q}_p$ in $G(\bar{A}^T)$ with $\overline{W} := \cup_{j=1}^{p} \overline{Q}_j = W \cup \{m+1, \ldots, m+\delta\}$. Then, the COQ inequality

$$\sum_{i \in \overline{W}} \sum_{b=1}^{\beta} x_i^b \leq \alpha(p, q, \beta, \kappa) \tag{5}$$

is valid for $P(\bar{A}, \beta, \kappa)$ and, of course, also for

$$P(\bar{A}, \beta, \kappa) \cap \{x_i^b = 0 : \ i = m+1, \ldots, m+\delta, \ b = 1, \ldots, \beta\}.$$

Since the artificial variables in this polytope attain only values of zero, this remains true if one sets their coefficients in (5) also to zero. But as this results in the desired extended COQ inequality (4) and

$$P(\bar{A}, \beta, \kappa) \cap \{x_i^b = 0 : \ i = m+1, \ldots, m+\delta, \ b = 1, \ldots, \beta\} = P(A, \beta, \kappa) \times \{0\}^{\delta \times \beta},$$

the theorem follows by a projection on the space of the original variables. □

The last *star inequality* that we present in this section is special in the sense that it is the only one with non-0/1 coefficients. It was designed to deal with rows with many neighbors.

3.13 Theorem (Star) *Let $G(A^T) = (V, E)$ and consider some row $i \in V$ with $|\gamma(i)| > \kappa$. Then the star inequality*

$$(|\gamma(i)| - \kappa + 1) \sum_{b=1}^{\beta} x_i^b + \sum_{j \in \gamma(i)} \sum_{b=1}^{\beta} x_j^b \leq |\gamma(i)|$$

is valid for $P(A, \beta, \kappa)$.

Proof. If i is assigned to some block b, then all rows in $\gamma(i)$ can only be assigned to b, but at most $\kappa - 1$ of them. The case where i is not assigned is trivial. $\square$

The star inequality can be viewed as a lifting of the (redundant) inequality

$$\sum_{j \in \gamma(i)} \sum_{b=1}^{\beta} x_j^{\beta} \leq |\gamma(i)|$$

and we want to close this section with another simple lifting theorem for block-invariant inequalities with 0/1 coefficients.

3.14 Theorem (Strengthening) *Let $G(A^T) = (V, E)$, W be a subset of V, and $\sum_{i \in W} \sum_{b=1}^{\beta} x_i^b \leq \alpha$ be a valid inequality for $P(A, \beta, \kappa)$. If for some row $j \notin W$ the condition*

$$|W \setminus \gamma(j)| + \kappa \leq \alpha$$

holds, then $\sum_{i \in W \cup \{j\}} \sum_{b=1}^{\beta} x_i^b \leq \alpha$ is also valid for $P(A, \beta, \kappa)$.

Proof. If j is assigned to some block b, the rows in $\{j\} \cup \gamma(j)$ can only be assigned to b, but at most κ of them. $\square$

4 A Branch-And-Cut Algorithm

The polyhedral investigations of the last section form the basis for the implementation of a branch-and-cut algorithm for the solution of the matrix decomposition problem. This section describes the four main ingredients of this code: Separation and LP-management, heuristics, problem reduction, and searchtree management.

4.1 Separation and LP-Management

We use all of the inequalities described in Section 3 as cutting planes in our algorithm. It will turn out that some of them appear in large numbers. We thus opted for the following separation strategy: We try to identify many inequalities using fast heuristics, but add only selected ones to the LP. More expensive separation methods are used depending on their success.

We classify our separation routines according to their basic algorithmic principles: Inspection (enumeration), greedy heuristics, other heuristics, exact polynomial methods, and "hybrid" methods (combinations of exact and heuristic methods).

The most simple methods are used for *big-edge*, *two-partition*, and *star inequalities*: These classes can be separated by simple *inspection*, the details for two-partition inequalities were already described in Section 3.

Clique, *z-clique*, and *z-cover inequalities* are separated using *greedy heuristics*. In the last two cases, these methods start by sorting the rows of A with respect to increasing z-value, i.e., such that $z(x)_{i_1} \geq z(x)_{i_2} \geq \cdots \geq z(x)_{i_m}$. Then the greedy heuristic is called m times, once for each row i_j. In each call, i_j is used to initialize a tree/clique with respect to $G(A^T)$, that is iteratively extended greedily in the order of the z-sorting of the rows until z_{i_j} becomes zero and the growing procedure stops. There is also a second variant for z-cliques that is an adaptation of a similar routine by Hoffman and Padberg [1993]. Here we call the greedy heuristic once for each column of A and initialize the clique with the support of this column. Having detected a violated clique inequality in one of these ways, we lift randomly determined additional rows with zero z-value sequentially into the inequality. This is done by applying the strengthening procedure of Theorem 3.14 which in this case amounts to a further growth of the clique by randomly determined rows of zero z-value. We tried to strengthen cover inequalities, but the computational effort was not justified by the one or two coefficients that were usually lifted. But, as was already mentioned in Section 3, we (heuristically) keep track of the connectivity of the growing graph. If the connectivity is 2 after the graph reached size $\kappa + 1$, we add another two-connected node if possible. Figure 2 gives more detailed pseudocode for z-cover separation. Separation of clique inequalities is done using exactly the same routine as for z-cliques, but applied to $G_c(A, \beta)$ with node weights given by the x-variables.

z-cover separation
Input: $z(x) \in \mathbb{R}^m$, $G(A^T)$
Output: Node set T of a one- or two-connected subgraph of $G(A^T)$

```
begin
    sort z(x) such that z(x)_{i_1} ≥ z(x)_{i_2} ≥ ··· ≥ z(x)_{i_m};
    for k := 1 to m
        T ← {i_k};
        connectivity ← 2;
        for j := 1 to m
            if j = k or γ(i_j) ∩ T = ∅   continue;
            if |T| = κ + 1 and |γ(i_j) ∩ T| = 1   continue;
            if |T| ≥ 2 and |γ(i_j) ∩ T| = 1   connectivity ← 1;
            T ← T ∪ {i_j};
            if |T| ≥ κ + connectivity   break;
        endfor;
    endfor;
    return T and connectivity;
end;
```

Figure 2: Separating z-cover inequalities with a greedy heuristic.

z-Cycle inequalities are separated in the following heuristic way. We look at some path P with end-nodes u and v, where initially u and v coincide. In each iteration we extend the path at one of its end-nodes by a neighbor w with maximal $z(x)_w$-value. Let j_w be a column of A that connects w to the path P. Since j_w forms a clique in $G(A^T)$ there are additional nodes that can be potentially added to the path if the support of j_w is greater than two, i. e., $|\operatorname{supp}(A_{\cdot j_w})| > 2$. We store these additional nodes in a buffer which will be exploited later in the heuristic. Now we test whether the new path P extended by w can be closed to a cycle C that satisfies $|C| > \kappa$ and $|C| \neq 0 \bmod (\kappa+1)$. This is done by looking for a column j of A that contains both end-nodes of P (one of them w). $\operatorname{supp}(A_{\cdot j})$ again forms a clique, and the additional nodes in this clique together with the nodes in the buffer give the flexibility to add further nodes to the cycle. This freedom is exploited in our routine. We try the procedure for several starting nodes $u = v$, whose number depends on the success of the heuristic.

Separation of the *composition of clique* inequalities is not easy: We do not even know a way to compute the right-hand side $\alpha(p, q, \beta, \kappa)$ in polynomial time! But there are problems in our test set, e.g., `pipex` (see Section 5.2), where compositions of cliques occur and there seems to be no way to solve this (small!) problem without them. Our heuristic was developed to capture these cases. It lead to the development of the more general class of extended COQ inequalities, which are easier to find. The idea is as follows.

Let us start with a composition of cliques $Q_1, \ldots, Q_p$ and $P_1, \ldots, P_q$ as stated in Theorem 3.11. Suppose that these cliques are contained in the columns $1, \ldots, p, p+1, \ldots, p+q$ of the matrix A, i.e., $\operatorname{supp}(A_{\cdot i}) \supseteq Q_i$, $i = 1, \ldots, p$, and $\operatorname{supp}(A_{\cdot i}) \supseteq P_i$, $i = p+1, \ldots, p+q$. Consider a *column/column-incidence matrix* S of A defined by

$$s_{ij} = \begin{cases} k, & \text{for } k \in \{l : a_{li} \neq 0 \neq a_{lj}\} \text{ arbitrary, but fixed;} \\ 0, & \text{if } A_{\cdot i}^T A_{\cdot j} = 0, \end{cases}$$

i.e., $s_{ij} = k \neq 0$ if and only if columns i and j intersect in some row k and in case there is no unique k we pick an arbitrary, but fixed one. Suppose for the moment that all entries in the submatrix $S_{\{1,\ldots,p\}\times\{p+1,\ldots,p+q\}}$ of S are mutually different, that is, there is no row index k that appears more than once. Then the composition of cliques corresponds to the rectangle submatrix $S_{\{1,\ldots,p\}\times\{p+1,\ldots,p+q\}}$ of S that is completely filled with nonzeros: The rows that appear on the left-hand side of the COQ inequality (2) are exactly those appearing in the matrix $S_{\{1,\ldots,p\}\times\{p+1,\ldots,p+q\}}$. In other words, the node set W in (2) is $W = \{s_{ij} : i = 1, \ldots, p, j = p+1, \ldots, p+q\}$. Thus, given some vector $x \in \mathbb{R}^{m\times\beta}$, the left-hand side of (2) is $\sum_{i=1}^{p}\sum_{j=p+1}^{p+q}\sum_{b=1}^{\beta} x^b_{s_{ij}}$ and a final calculation of the right-hand side allows to check for a possible violation of the inequality.

Our heuristic tries to go the reverse direction: It identifies large filled rectangles in S and derives COQ and xCOQ inequalities from them. There are three difficulties. First, a clique in a composition can not only be a subset of a column of A, but any clique in $G(A^T)$. However, we have not incorporated this generality in our heuristic, because we do not know how to select a promising set of cliques in $G(A^T)$. Second, columns in A that form a composition of cliques may not appear in the right order: The rectangle identifies itself only after a suitable permutation of S. In this case, we have to reorder the columns and rows of S. We obtain a filled rectangle submatrix $S_{I\times J}$ of S by starting with each of column j of S once, extend this $1\times|\operatorname{supp}(A_{\cdot j})|$ rectangle submatrix by columns that fit best in a greedy way, sort its rows lexicographically, and consider all maximal filled submatrices in the upper left corner as potential COQ-rectangles. A third serious problem arises when two columns of A intersect in more than one row. In this case the entries of the matrix S are no longer uniquely determined and it can happen that the entries of the rectangular submatrix $S_{I\times J}$ under consideration are no longer mutually different. Then $S_{I\times J}$ corresponds no longer to a composition of cliques and the inequality $\sum_{ij\in I\times J}\sum_{b=1}^{\beta} x^b_{s_{ij}} \le \alpha\bigl(|I|,|J|,\beta,\kappa\bigr)$ is in general not valid. But one can set duplicate entries in S to zero until, for every row k, there is only one representative $s_{ij}=k$ left; denote the resulting matrix by S'. Then the sets

$$\overline{Q}_i := \{s'_{ij} : s'_{ij} \ne 0,\ j\in J\},\quad i\in I \qquad\text{and}\qquad \overline{P}_j := \{s'_{ij} : s'_{ij}\ne 0,\ i\in I\},\quad j\in J$$

of non-zero entries in the rows and columns of S' form an extended composition of cliques and the corresponding xCOQ inequality

$$\sum_{k\in\operatorname{im} S_{I\times J}}\sum_{b=1}^{\beta} x^b_k \le \alpha(|I|,|J|,\beta,\kappa)$$

is valid for $P(A,\beta,\kappa)$, where $\operatorname{im} S_{I\times J}=\{s_{ij} : ij\in I\times J\}$ denotes the set of row indices that appear in the submatrix $S_{I\times J}$. The interesting feature of separating extended COQ inequalities instead of COQs is that the generalization gives us the algorithmic freedom to handle multiple occurrences of rows in filled rectangles of S and this is the key to a successful heuristic separation of an otherwise rigid structure. The price for this, of course, is a reduced support in the left-hand side. To pay this price only when necessary, we heuristically determine a column/column-intersection matrix S with a large variety of rows in $\operatorname{im} S$. The right-hand side itself is computed in amortized (pseudo-polynomial) time of $O(\beta\kappa n^2)$ steps by a dynamic program (for our tests $\beta\le 4$ and $\kappa=O(n)$, and thus this effectively amounts to $O(n^3)$).

The reader might have noticed that several degrees of freedom in this separation routine can be used to search for rectangles with large z-value and this is what we would like to find. However, the running time of the method is too large to apply it after each LP and when we did, we did not find additional cuts. We thus call the routine only once, determine some promising COQs by combinatorial criteria, store them in memory, and separate them by inspection.

To separate *clique inequalities* (for $\beta>2$), we use an *exact branch-and-bound algorithm* for the maximum weight clique problem. Although in principle exponential, this algorithm works fast for the separation problems coming up in our matrix decomposition instances because the maximum clique size is bounded by β. We have also tried to separate z-cliques exactly, but we never observed that additional cuts were found: In the small examples, the greedy heuristic is good enough, while in the larger ones with high capacities cliques of size κ don't seem to exist. Another exact, but this time polynomial, algorithm is used to separate *cycle inequalities*: We apply the odd-cycle algorithm described in Lemma 9.1.11 in Grötschel, Lovász, and Schrijver [1988].

Finally, a mixture of exact and heuristic ideas is used in a hybrid algorithm to separate the *bin-packing inequalities.* We start by determining a node set W that can result in a violated inequality. A necessary condition for this is

$$\sum_{i\in W} z(x)_i > |W|-1 \iff 1 > \sum_{i\in W}(1-z(x)_i)$$

and it is reasonable to construct W by iteratively adding rows that have a z-value close to one. We thus sort the nodes with respect to increasing z-value and add them to W in this order as long as the condition stated above is satisfied. This node set W induces a subgraph $(W,E(W))$ of $G(A^T)$ and we determine the components of this subgraph. The resulting bin-packing problem (see page 9) is solved using an exact dynamic programming algorithm (with a time bound).

In addition to these classical types of cutting planes we also use a number of *"tie-breaking" inequalities* to cut off decompositions that are identical up to block permutations or give rise to multiple optima for other reasons as a means to counter dual degeneracy and stalling. These inequalities are in general not valid for

$P(A, \beta, \kappa)$, but for at least one optimal solution. The most simple kind of these cuts are the *permutation inequalities*

$$\sum_{i=1}^{m} x_i^b \leq \sum_{i=1}^{m} x_i^{b+1}, \quad b = 1, \ldots, \beta - 1,$$

stating that blocks with higher indices are of larger size. To break further ties, we supplement them with inequalities stipulating that in case of equal sized blocks the row with the smallest index will be assigned to the block with smaller index. These *strengthened permutation inequalities* read

$$x_k^{b+1} + \sum_{i=1}^{m} x_i^b - \sum_{i=1}^{m} x_i^{b+1} \leq \sum_{i=0}^{k-1} x_i^b, \quad b = 1, \ldots, \beta - 1, \ k = 2, \ldots, m - 1.$$

If $\sum_{i=1}^m x_i^b - \sum_{i=1}^m x_i^{b+1} < 0$, the inequality is redundant, but in case of equality, the row with the smallest index in blocks b and $b+1$ must be in block b. The case $k = m$ is left out because it yields a redundant inequality. Both permutation and strengthened permutation inequalities can be separated by *inspection*.

Another idea that we use to eliminate multiple optima is based on the concept of *row preference*. We say that row i is *preferred* to row j or, in symbols, $i \prec j$ if

$$\gamma(i) \subseteq \gamma(j)$$

with respect to the row intersection graph $G(A^T)$. We may in this situation not know whether or not row i or j can be assigned to a block in some optimal solution, but we can say that for any decomposition x with $z(x)_j = 1$, say $x_j^b = 1$, either $z(x)_i = 1$ or we can get a feasible decomposition $x' = x - e_j^b + e_i^b$ with the same number of rows assigned. In this sense, row i is more attractive than row j. If we break ties on row preference by indices (i.e., $i \prec j \iff \gamma(i) \subsetneq \gamma(j) \vee (\gamma(i) = \gamma(j) \wedge i < j)$), row preferences induce a partial order that we represent in a transitive and acyclic digraph

$$D(A) := (V, \{(i,j) : i \prec j\}).$$

Since the number of row preferences tends to be quadratic in the number of rows, we thin out this digraph by removing all transitive (or implied) preferences. The remaining row preferences are forced in our code by adding the *row preference inequalities*

$$\sum_{b=1}^{\beta} x_i^b \geq \sum_{b=1}^{\beta} x_j^b \quad \text{for } (i,j) \text{ with } i \prec j.$$

These can sometimes be strengthened to

$$x_i^b \geq x_j^b \quad \text{for all } b = 1, \ldots, \beta,$$

if we can be sure that rows i and j can not be assigned to different blocks in any decomposition. This will be the case, for example, if i and j are adjacent in $G(A^T) = (V, E)$ or if both i and j are adjacent to some third row k preferable to both of them (i.e., $i \prec j$, $k \prec i$, $k \prec j$, $ik \in E$ and $jk \in E$). Once $D(A)$ is set up, row preference inequalities can be separated by *inspection*.

Our last separation routine uses a *cut pool* that stores all inequalities found by the hitherto explained algorithms: The *pool separation* routine just checks all inequalities in the pool for possible violation.

The separation algorithms described in the previous paragraphs turned out to be very successful: Not only block-discernible (permutable) inequalities like two-partitions are found in large numbers, also block-invariant cuts like z-covers occur in abundance. Controlling the growth of the LP-relaxation is thus the main goal of our *separation and LP-maintenance strategy*. We start with a minimal LP-relaxation containing (besides the bounds) only the block assignment and block capacity constraints plus the $\beta - 1$ permutation inequalities. The cuts that are separated by inspection, i.e., big-edge inequalities, star inequalities, tie-breaking inequalities, and composition of clique inequalities are placed in a *cut pool*; they will be found by pool separation. The separation algorithms are called dynamically throughout the course of the branch-and-cut algorithm. After an LP is solved, we call the pool separation routine, followed by two-partition inequality separation and a couple of heuristics: The z-cover heuristic is called as it is, but application of the more expensive z-clique and z-cycle algorithms is controlled by a simple time- and success-evaluation. This control mechanism is motivated by the observation that our test set fell into two groups of examples, where one of these routines was either indispensable or essentially did not find a single cut. We empirically try to adapt to these situations by calling the separation routines only if their past success is proportional to the running time, or more precisely, if after the first call

$$\frac{\#\text{ of successful calls} + 1}{\#\text{ of calls}} > \frac{\text{time spent in routine}}{\text{total time spent in separation}}.$$

A call is counted as successful if a violated cut is found. If $\beta > 2$, there can be clique inequalities that are not two-partition constraints and in this case we next call the exact clique separation routine, that returns at most one cut. The branch-and-bound algorithm used there turned out to be fast enough to be called without any further considerations. Finally, we separate bin-packing inequalities. To avoid excessive running times due to the dynamic program, the routine is called with a time limit: The dynamic program will be stopped if the time spent in bin-packing separation exceeds the cumulated separation time of all other separation routines.

All violated cuts determined in this separation process are not *added* directly to the LP-relaxation, but stored in a *cut buffer* first. This buffer is saved to the pool, and then a couple of promising cuts are selected to strengthen the LP-relaxation. Our criteria here have an eye on the amount of violation and on the variety of the cuts. Since inequalities of one particular type tend to have similar support, we restrict the number of cuts per type and prefer to add inequalities of other types, even if they are not among the most violated. To accomplish this we add the

$$\frac{\#\text{ of cuts in cut buffer}}{\#\text{ number of types of cuts}}$$

most violated cuts of each type to the LP-relaxation. We also *delete* cuts from the LP-relaxation if they become non-binding by a slack of at least 10^{-3}, but keep them in the cut pool for a possible later pool separation.

Another feature of our code that aims for small LPs is to *locally setup* the LP-relaxation prior to computation at any node of the searchtree. This means that when branching on some node v we store at each of its sons a description of the last LP solved at v and of the optimal basis obtained. When we start to process v's sons, we set up this LP from scratch and load the associated (dual feasible) basis. In this way, we continue the computation exactly at the point where it stopped and the LP will be the result of a contiguous process independent of the node selection strategy. We have compared this approach to one where the start-LP at each newly selected node is just the last LP in memory and this leads to larger LPs and larger running times.

While these strategies were sufficient to keep the size of the LP-relaxation under control, explosive growth of the cut pool was a serious problem in our computations until we implemented the following *cut pool management*. We distinguish between disposable and indisposable cuts in the pool. *Indisposable cuts* are inequalities that are needed to set up the LP-relaxation at some node in the searchtree yet to be processed and all big-edge, star, and tie-breaking inequalities. All other cuts are *disposable* and can potentially be deleted from the cut pool, possibly having to be recomputed later. In order to control the pool size we restrict the number of disposable cuts in the pool by eliminating cuts that have not been in any LP for a certain number of iterations. This number depends on the size of the pool and the ratio of disposable to indisposable cuts.

The LPs themselves are solved with the `CPLEX 4.0` dual simplex algorithm using steepest edge pricing, see the `CPLEX` documentation CPLEX [1997].

4.2 Heuristics

Raising the lower bound using cutting planes is one important aspect in a branch-and-cut algorithm, finding good feasible solutions early to enable fathoming of branches of the searchtree is another and we have implemented several primal heuristics for our matrix decomposition code. Since different nodes in a branch-and-bound tree correspond to different fixings of variables to zero or one, the heuristics should respect these fixings to increase the probability of finding different solutions. Applied at the root node where (at least initially) no variables are fixed, our methods can be seen as LP-based or pure combinatorial heuristics for the matrix decomposition problem.

Our heuristics fall into three groups: "Primal" methods that iteratively fix block assignments, "dual" methods that iteratively exclude assignments until decisions become mandatory due to lack of alternatives, and an improvement method that is applied as an "afterburner" to enhance the quality of the two groups of opening heuristics.

The *primal methods* consist of a *greedy algorithm* and a *bin-packing heuristic*, both are LP-based. The greedy algorithm starts by ordering the x_i^b variables; with probability $\frac{1}{2}$ a random ordering is chosen, otherwise a sorting according to increasing x-value is used. The rows are assigned greedily to the blocks in this order. This heuristic is similar in spirit to the popular "LP-plunging" method, i.e., the iterative rounding of some fractional LP-value to an integer followed by an LP-reoptimization, but much faster. We have also tried LP-plunging, but for the matrix decomposition problem the results were not better than with the simple greedy method, while the running time was much larger. The bin-packing heuristic starts by determining a set of nodes W that will be assigned to the blocks and used to set up a corresponding bin-packing problem. In order to find a better decomposition than the currently best known with, say, $z^\star$ rows assigned, W should be of cardinality at least $z^\star + 1$ and therefore we take the $z^\star + 1$ rows with the largest $z(x)$-values to be the members of W. The corresponding bin-packing problem is set up and solved with the same dynamic program that we used for the separation of the bin-packing inequalities; it is also called with a time limit, namely 10 times as much as all other primal heuristics (that are very fast) together. Clearly, we also watch out for better solutions that might be detected in bin-packing separation.

The *dual methods* also respect variable fixings, but are not LP-based. The idea behind them is not to assign rows to blocks, but to iteratively eliminate assignments of "bad" rows. Suppose that a decision was made to assign certain rows (assigned rows) to certain blocks, to exclude other rows from assignment (unassigned rows), while for the remaining rows a decision has yet to be made (free rows). Removing the unassigned nodes from the row intersection graph $G(A^T)$ leaves us with a number of connected components, some of them larger than the maximum block capacity κ, some smaller. Both variants of the dual method will break up the components that are larger than the block capacity κ by unassigning free rows until no more such components exist. At this point, a simple first-fit decreasing heuristic is called to solve the corresponding bin-packing problem. The two variants differ in the choice of the next bad row to remove. Variant I chooses the free row in some component of size larger than κ with the largest degree with respect to the row intersection graph $G(A^T)$, variant II excludes assignment of a free row of some component with size larger than κ with the largest indegree with respect to $D(A)$, or, in other words, the least preferable row. We have also tried to use a dynamic program to solve the bin-packing problems, but it did not provide better results in our tests.

Our *improvement heuristic* is a variation of a local search technique presented by Fiduccia and Mattheyses [1982]. Given some block decomposition, it performs a sequence of local exchange steps each of the following type. Some assigned row is chosen to be made unassigned opening up possibilities to assign its unassigned neighbors. These assignments are checked and feasible assignments are executed. The details are as follows. The heuristic performs a number of passes (10 in our implementation). At the beginning of each pass, all rows are eligible for unassignment in the basic exchange step. Each row may be selected only once for unassignment in each pass and will then be "locked". Candidates for becoming unassigned are all currently assigned and unlocked rows. These candidates are rated according to the number of possible new assignments (computed heuristically) and we choose the one that is best with respect to this rating. As a special annealing-like feature, the algorithm will also perform the exchange step if it leads to a change of the current solution to the worse. If no exchange step is possible because all assigned rows are already locked, the pass ends and the next pass is started.

The *strategy to call the heuristics* is as follows. The primal methods are called after each individual LP, whereas the dual heuristics are called only once at each node in the branch-and-bound tree, because they behave in a different way only due to changes in the variable fixings.

4.3 Problem Reduction

We use a couple of problem reduction techniques to eliminate redundant data. First we apply an *initial preprocessing* to the matrix A before the branch-and-cut algorithm is initiated. The purpose of this preprocessing step is to eliminate columns from the matrix A without changing the row intersection graph. We first perform a couple of straightforward tests to identify columns that are contained in other columns and can thus be deleted: We remove empty columns, then unit columns, duplicate columns, and finally by enumeration columns that are contained in others.

These simple initial preprocessing steps are amazingly effective as we will see in the section on computational results. In principle, the number of rows can be reduced also. For example, empty rows could be eliminated and later used to fill excess capacity in any block, duplicate rows or rows with just one nonzero entry could be eliminated by increasing the capacity requirements of one of its adjacent rows. These reductions, however, lead to changes in the IP model and affect all separation routines discussed so far so that we refrained from implementing them.

In addition to this initial preprocessing we do *local fixings* at the individual nodes of the branch-and-bound searchtree after each call to the LP-solver. Apart from *reduced cost fixing* and *fixing by logical implication* (i.e., if x_i^b is fixed to one, $x_i^{b'}$ will be fixed to zero for all blocks $b' \neq b$), we try to identify rows that cannot be assigned to any block given the current state of fixings. To this purpose we look at all rows W that are currently fixed for assignment to some block. We then check for each unassigned row i whether the subgraph $(W \cup \{i\}, E(W \cup \{i\}))$ of $G(A^T) = (V, E)$ contains a component with more than κ rows. If so, row i can be fixed to be unassigned.

4.4 Searchtree Management

Despite our efforts to understand the polyhedral combinatorics of the matrix decomposition problem, we do not have a strong grip on the corresponding polytope and after an initial phase of rapid growth of the lower bound, stalling occurs in the presence of significant duality gaps. We believe that —up to a certain point— it is favorable in this situation to resort to branching early, even if there is still slow progress in the cutting plane loop. In fact, we apply a rather "aggressive" *branching strategy*, splitting the currently processed node if the duality gap could not be reduced by at least 10% in any 4 consecutive LPs. On the other hand, we *pause* a node (put it back into the list of nodes yet to be processed) if the local lower bound exceeds the global lower bound by at least 10%.

Branching itself is guided by the fractional LP-values. We first look for a most fractional $z(x)$-value. If, e.g., $z(x)_i$ is closest to 0.5 (breaking ties arbitrarily), we create $\beta + 1$ new nodes corresponding to the variable fixings

$$x_i^1 = 1, x_i^2 = 1, \ldots, x_i^\beta = 1, \quad \text{and} \quad \sum_{b=1}^{\beta} x_i^b = 0.$$

In other words, we branch on the block assignment constraint corresponding to row i. The advantage of this scheme is that it leads to only $\beta + 1$ new nodes instead of 2^β-nodes in an equivalent binary searchtree. If all $z(x)$-values are integral, we identify a row with a most fractional x-variable and perform the same branching step. We have also tried other branching rules by taking, for instance, the degree of a row in $G(A^T)$ into account, but the performance was inferior to the current scheme.

5 Computational Results

In this section we report on computational experiences with our branch-and-cut algorithm for the solution of matrix decomposition problems arising in linear and integer programming problems. Our aim is to find answers to two complexes of *questions*. First, we would like to evaluate our branch-and-cut approach: What are the limits in terms of the size of the matrices that we can solve with our algorithm? What is the quality of the cuts, do they provide a reasonable solution guarantee? Second, we want to discuss our concept of decomposition into bordered block diagonal form: Do the test instances have this structure or are most integer programming matrices not decomposable in this way? Does knowledge of the decomposition help solving them? And do our heuristics provide reasonable decompositions that could be used within a parallel LU-factorization framework or an integer programming code?

Our *test set* consists of matrices from real-world linear programs from the `Netlib` and integer programming matrices from the `Miplib`. In addition, we consider two sets of problems with "extreme" characteristics as benchmark problems: Some small matrices arising from Steiner-tree packing problems, see Grötschel, Martin, and Weismantel [1996], and equipartition problems introduced in Nicoloso and Nobili [1992]. The Steiner-tree problems are known to be in bordered block diagonal form and we wanted to see whether our code is able to discover this structure. The equipartition problems, on the other hand, are randomly generated. Our complete test data is available via anonymous `ftp` from `ftp.zib.de` at `/pub/Packages/mp-testdata/madlib` or can be retrieved alternatively via world wide web at URL `ftp://ftp.zib.de/pub/Packages/mp-testdata/madlib/index.html`.

In the following subsections, we report the *results of our computations* on the different sets of problems.

Our algorithm is implemented in `C` and consists of about 36,000 lines of code. The test runs were performed on a Sun Ultra Sparc 1 Model 170E and we used a time limit of 1,800 CPU seconds. The format of the upcoming tables is as follows: Column 1 provides the name of the problem, Columns 2 to 4 contain the number of rows, columns and nonzeros of the matrix to be decomposed. The two succeeding columns give the number of columns and nonzeros after presolve. Comparing Column 3 with 5 and 4 with 6 shows the performance of our preprocessing. The succeeding 5 columns give statistics about the number of cuts generated by our code. There are, from left to right, the number of initial cuts (*Init*) including block assignment, block capacity, big-edge, star, and tie-breaking inequalities (but not composition of clique inequalities, although they are also separated from the pool), the number of z-cover (*Cov*), the number of two-partition (*2part*), the sum of the number of bin-packing, cycle, z-cycle, clique, z-clique, and composition of clique inequalities (*BCC*), and finally the number of violated inequalities separated from the pool (*pool*). The following two columns (Columns 12 and 13) show the number of branch-and-bound nodes (*Nod*) and the number of LPs (*Iter*) solved by the algorithm. The next eight columns give solution values. We do not report the number of assigned rows, but the number of rows in the border, because it is easier to see whether the matrix could be decomposed into block diagonal form (in this case the value is zero) or close to this form (then the value is a small positive integer). *Lb* gives the global lower bound provided by the algorithm. It coincides with the value of the upper bound *Ub* (next column) when the problem is solved to *proven optimality*. Skipping two columns for a moment, the next four columns refer to the heuristics. *G*, *D1*, *D2* and *B* stand for the *greedy*, the *dual (variant I and II),* and the *bin-packing* heuristic. The corresponding columns show the best solutions obtained by these heuristics throughout the computations at the root node. If this value coincides with the number of rows of the matrix, all rows are in the border and the heuristic failed. The two (skipped) columns right after *Ub* show which heuristic *He* found the best solution after *No* many branch-and-bound nodes (1 means it was found in the root node, 0 means that preprocessing solved the problem). The additional letter *I* indicates that the value was obtained by a succeeding call to the *improvement heuristic*. *BS* means that the bin-packing separation routine found the best solution, an asterisk $*$ shows that the LP solution provided an optimal block decomposition. The remaining five columns show timings. The last of these columns *Tot* gives the total running time measured in CPU seconds. The first four columns show the percentage of the total time spent in cut-management (*Cm*), i.e., local setup, LP-, cut-buffer, and pool-management, the time to solve the linear programs (*LP*), the time of the separation algorithms (*Sep*), and the time for the heuristics (*Heu*).

5.1 The `Netlib` Problems

The first test set that we are going to consider consists of matrices that arise from *linear programming problems* taken from the `Netlib`*. We investigated whether *basis matrices* corresponding to optimal solutions of these linear programs can be decomposed into (bordered) block diagonal form. These bases were taken from the dual simplex algorithm of CPLEX [1997]. Analyzing the decomposibility of such matrices gives insight into the potential usefulness of parallel LU-factorization methods within a simplex-type solver. In this context β reflects the number of processors that are available. We have chosen $\beta = 4$, since this is somehow the first interesting case where parallelization might pay. As a heuristic means for good load balancing we aim at equal-sized blocks and have set the capacity to $\kappa := \frac{\#rows}{4}$ rounded up. We tested all instances with up to 1,000 rows.

Table 1 shows the *results* of these experiments. The problems up to 100 rows are easy. The range of 100–200 rows is where the limits of our code become visible and this is the most interesting "hot area" of our table: The problems here are already difficult, but because of the combinatorial structure and not because of sheer size. The results for the problems with more than 200 rows are of limited significance, because these matrix decomposition problems are large-scale and the algorithm solves too few LPs within the given time limit. We can only solve a couple of readily decomposable large instances, but it is worth noticing that a significant number of such instances exists: The difficulty of matrix decomposition problems depends as much on the structure of the matrix as on the number of rows, columns, or nonzeros.

Let us first investigate the "*dual side*" of the results. We observe that we solve very few problems at the root node (only 9 out of 77), and that the number of cuts is very large, in particular in the hot area of the table. The reason for this is basically the symmetry of the problem, as can be seen from the pool separation column (*Pool*) that counts, in particular, all violated tie-breaking cuts. Unfortunately, we don't see a way to get around this phenomenon, but we believe that the symmetry mainly prevents us from solving difficult instances of larger size. The quality of the cuts is in our opinion reasonable, as can be seen

*Available by anonymous `ftp` from `ftp://netlib2.cs.utk.edu/lp/data`.

Name	Original			Presolved		Cuts					B&B		Best Solutions				Heuristics at Root				Time				
	rows	col	nz	col	nz	Init	Cov	2part	BCC	Pool	Nod	Iter	Lb	Ub	He	No	G	D1	D2	B	Cm	LP	Sep	Heu	Tot
seba	2	2	2	0	0	0	0	0	0	0	1	0	0	**0**	-	0	-	-	-	-	0%	0%	0%	0%	0.0
afiro	20	20	34	10	24	106	12	6	5	2	1	2	2	**2**	B	1	3	3	3	2	0%	40%	10%	0%	0.1
fit1d	24	24	178	9	126	298	97	38	265	105	6	18	16	**16**	B	1	16	17	17	16	6%	50%	25%	6%	0.7
fit2d	25	25	264	11	147	379	549	458	1504	892	41	99	18	**18**	D1	1	20	18	18	25	12%	54%	22%	5%	5.5
sc50b	28	28	84	27	82	124	2304	1291	152	5452	56	182	11	**11**	IG	1	11	11	11	28	6%	71%	12%	4%	12.7
sc50a	29	29	88	24	77	137	1062	654	53	1487	26	79	8	**8**	ID1	1	8	8	8	29	3%	78%	11%	2%	6.8
kb2	39	39	213	16	164	328	38	21	2	8	1	3	14	**14**	IG	1	14	14	14	39	4%	69%	14%	2%	0.5
vtpbase	51	51	198	27	149	415	9130	4796	407	32630	561	1025	14	**14**	IG	1	14	17	17	15	7%	57%	20%	8%	94.6
bore3d	52	52	311	29	262	579	2241	1025	2333	2706	51	116	23	**23**	D1	9	24	24	24	52	7%	69%	17%	2%	24.0
adlittle	53	53	203	35	167	454	818	244	20	515	11	31	10	**10**	D1	1	10	10	10	53	6%	72%	15%	2%	5.0
blend	54	54	313	28	274	547	3357	1553	2757	6143	61	166	20	**20**	IG	2	21	25	25	21	6%	66%	20%	3%	35.9
recipe	55	55	100	24	69	317	0	14	3	2	1	3	1	**1**	IB	1	3	4	4	1	0%	51%	18%	10%	0.4
scagr7	58	58	242	34	210	439	770	331	718	468	16	30	21	**21**	B	4	25	25	25	58	6%	46%	32%	10%	5.2
sc105	59	59	220	48	203	318	6200	3415	151	19447	66	207	16	**16**	B	20	17	17	17	59	4%	82%	9%	2%	85.1
stocfor1	62	62	180	25	143	497	2395	801	55	2977	21	65	10	**10**	B	7	12	12	12	62	6%	68%	17%	4%	15.2
scsd1	77	77	215	70	207	500	4264	1570	60	6331	21	74	8	**8**	IB	5	10	10	10	77	3%	81%	10%	3%	53.1
beaconfd	90	90	618	48	576	1626	4618	1403	3231	6296	121	263	26	**26**	D1	21	28	28	28	90	4%	67%	21%	4%	115.7
share2b	93	93	482	37	340	825	7335	4636	212	14713	626	860	9	**9**	ID1	5	10	10	10	93	4%	77%	8%	6%	282.5
share1b	102	102	485	56	364	721	3769	2135	294	8204	806	1142	7	**7**	B	4	8	8	8	102	5%	65%	13%	9%	203.6
forplan	104	104	575	70	535	815	43011	16754	4783	145556	316	681	31	36	ID1	18	39	44	44	104	2%	85%	8%	3%	1801.1
scorpion	105	105	383	46	300	812	48144	12649	1178	113261	1206	2485	11	**11**	B	7	14	14	14	105	4%	67%	14%	9%	940.7
sc205	113	113	691	104	676	561	24727	15842	272	116813	136	299	28	39	IG	9	45	45	45	113	1%	92%	4%	1%	1806.6
brandy	113	113	874	83	774	1270	33012	13041	5037	107955	271	664	34	**34**	*	214	51	51	51	113	2%	82%	11%	2%	1200.7
lotfi	122	122	349	60	267	1026	85763	17243	1110	203255	1076	2217	19	**19**	B	279	23	23	23	122	4%	76%	12%	4%	1643.3
boeing2	122	122	435	70	383	1038	18823	5794	200	41101	71	237	19	**19**	B	33	31	32	32	122	2%	84%	7%	3%	411.9
tuff	137	137	820	84	748	1795	17387	4823	200	32293	116	306	26	**26**	B	43	28	35	35	137	2%	85%	7%	3%	635.3
grow7	140	140	1660	51	243	948	33529	12866	320	182151	206	424	10	18	IG	9	22	22	22	140	3%	80%	9%	6%	1801.2
scsd6	147	147	383	138	373	804	41055	12440	297	116468	111	331	10	22	IG	4	23	23	23	147	1%	87%	5%	4%	1805.1
e226	148	148	954	88	807	1469	33011	14063	272	112223	206	355	22	30	IG	15	44	44	44	148	1%	84%	7%	5%	1800.5
israel	163	163	1321	37	1014	2217	537	289	2115	237	11	25	98	**98**	IG	2	101	119	118	163	4%	32%	51%	10%	42.1
agg	164	164	669	52	551	1492	37378	12257	325	174665	341	539	19	24	IG	56	43	43	43	164	3%	81%	9%	5%	1802.3
capri	166	166	826	102	718	1625	9291	4015	396	9008	71	94	23	53	D1	7	59	64	64	166	0%	90%	4%	3%	1801.8
wood1p	171	171	2393	55	1340	1650	6253	2457	56	7758	46	94	28	**28**	B	29	32	32	32	171	2%	73%	15%	7%	277.6
bandm	180	180	1064	90	815	1690	11854	6042	405	31838	66	96	19	43	IG	11	58	58	58	180	1%	93%	4%	1%	1835.0
scrs8	181	181	887	110	675	1400	11399	5375	72	30637	56	85	17	43	IG	9	56	56	56	181	0%	92%	3%	2%	1802.1
ship04s	213	213	573	176	536	1530	1084	72	7	190	1	7	4	**4**	IB	1	8	8	8	4	5%	55%	31%	2%	10.8
scagr25	221	221	1627	91	1470	1831	14509	8239	6884	23224	121	261	62	69	B	53	81	81	81	221	1%	52%	21%	23%	1814.7
scfxm1	242	242	1064	154	922	2182	16607	3887	68	38232	31	80	10	30	IG	7	50	50	50	242	1%	87%	5%	5%	1803.7
stair	246	246	3402	216	3154	1552	7192	3058	33	1005	51	44	58	122	IG	3	123	133	133	246	0%	83%	10%	5%	1914.6
shell	252	252	493	235	476	1495	14086	1311	85	5878	46	101	4	**4**	B	17	8	8	8	252	3%	58%	16%	18%	187.8
standata	258	258	513	109	364	1733	0	0	0	0	1	1	1	**1**	D1	1	17	1	1	258	0%	52%	35%	3%	5.3
sctap1	269	269	640	77	373	1957	25028	4187	110	37924	61	143	9	19	D1	13	20	20	20	269	1%	86%	6%	5%	1801.2
agg2	280	280	1468	109	1275	2270	9856	4124	34	13961	31	45	13	78	IG	5	110	112	112	280	0%	89%	5%	3%	1810.8
agg3	282	282	1444	99	1162	2259	9572	4674	32	13575	26	43	12	79	IG	6	92	116	116	282	0%	91%	4%	2%	1808.6
ship08s	284	284	699	201	616	1909	7352	480	33	3403	11	35	4	**4**	B	3	10	10	10	284	3%	76%	14%	4%	144.0
boeing1	284	284	1384	174	1271	2670	13291	4198	48	22032	31	56	12	55	IG	2	56	56	56	284	1%	91%	3%	4%	1812.6
grow15	300	300	3680	102	489	2044	17134	5225	72	28665	21	77	7	20	ID1	1	20	20	20	300	1%	84%	5%	8%	1817.1
fffff800	306	306	1382	182	1237	3045	9590	3320	30	15509	36	51	12	52	IG	6	79	104	104	306	0%	89%	5%	3%	1820.2
etamacro	307	307	1005	215	907	1824	8431	3985	33	10512	16	36	6	83	IG	3	91	91	91	307	0%	93%	2%	2%	1832.6
ship04l	313	313	868	274	829	2321	4387	520	125	1918	356	491	5	**5**	B	14	8	8	8	313	1%	66%	3%	25%	1727.2
gfrdpnc	322	322	623	276	576	1569	13308	2328	53	4899	31	59	4	**4**	BS	7	8	8	8	322	2%	65%	9%	21%	398.4
ship12s	344	344	858	247	761	2313	1676	108	7	62	6	8	3	**3**	IB	2	10	10	10	344	4%	61%	24%	6%	35.4
finnis	350	350	831	178	653	2371	13570	3020	55	17997	26	61	7	27	B	4	31	31	31	350	1%	90%	4%	3%	1842.8
pilot4	352	352	3157	265	2988	3014	8563	4104	25	13824	16	29	6	106	IG	1	106	109	109	352	0%	87%	6%	4%	1881.6
standmps	360	360	836	217	691	2785	32633	4731	183	37695	71	210	7	10	D1	4	11	11	11	360	2%	79%	9%	7%	1802.0
degen2	382	382	2440	262	2230	2717	6922	1959	16	2396	26	25	12	114	ID1	1	114	114	114	382	0%	92%	3%	2%	1922.1
scsd8	397	397	1113	394	1109	1755	9128	4799	24	14671	6	25	4	63	IG	2	85	98	89	397	1%	90%	2%	5%	1912.1
grow22	440	440	5272	161	732	2984	10976	1880	26	10062	11	28	4	24	ID1	3	25	31	25	440	1%	83%	4%	11%	1808.9
bnl1	448	448	1656	307	1481	3788	6984	1623	16	6521	11	19	6	68	D1	1	68	68	68	448	0%	93%	2%	3%	2046.8
czprob	475	475	939	464	928	3640	1227	25	4	9	1	4	3	**3**	D1	1	3	3	3	475	4%	53%	35%	1%	42.0
scfxm2	485	485	2179	313	1906	4317	7582	1712	17	10240	6	18	5	65	IG	2	113	120	115	485	1%	80%	4%	13%	1849.6
perold	500	500	3277	424	3077	3448	5983	3669	13	6810	6	14	5	166	D1	1	166	166	166	500	0%	90%	3%	4%	2079.5
ship08l	520	520	1404	436	1320	3767	11855	703	76	4080	71	129	4	5	D1	9	12	12	12	520	1%	48%	7%	40%	1807.1
maros	545	545	2637	301	2323	4384	7750	2718	15	2539	11	18	8	103	D1	1	103	103	103	545	1%	85%	4%	8%	1832.8
ganges	576	576	3002	466	2892	4098	19490	2429	40	10251	16	42	6	15	IB	3	36	39	39	576	2%	65%	10%	21%	1805.4
pilotwe	613	613	2982	561	2833	3804	5489	4184	10	4238	6	11	3	181	IG	2	182	186	182	613	0%	83%	3%	11%	1911.8
nesm	622	622	1925	419	1461	3720	5929	3407	11	4785	6	12	3	151	D1	2	152	152	152	622	1%	86%	4%	7%	1803.8
fit1p	627	627	4992	1	627	5013	0	0	0	0	1	1	470	**470**	IG	1	470	470	470	627	0%	92%	5%	0%	726.4
25fv47	677	677	3750	442	3278	5699	4644	1315	7	482	6	9	4	190	IG	1	190	195	195	677	0%	90%	3%	5%	2102.4
ship12l	686	686	1883	589	1786	5026	3212	127	6	74	1	6	3	**3**	B	1	4	18	18	3	3%	57%	27%	8%	169.8
woodw	711	711	3044	528	2849	4517	3736	2395	6	399	11	9	5	184	IG	2	186	187	187	711	0%	89%	4%	5%	2069.4
scfxm3	728	728	3285	469	2876	6411	4969	1203	8	3541	6	9	3	104	D1	2	105	105	105	728	0%	90%	3%	4%	1924.6
pilotja	745	745	4738	548	4121	5708	4462	2129	7	670	6	8	3	226	ID1	1	226	226	226	745	0%	88%	4%	6%	1989.3
pilotnov	783	783	4428	498	3606	5479	3859	2466	6	1159	6	7	3	250	D1	1	250	250	250	783	0%	85%	3%	9%	1855.4
bnl2	940	940	3284	489	2509	6981	4930	880	7	634	6	8	3	199	D1	1	199	199	199	940	0%	81%	4%	11%	2051.2
sctap2	977	977	1491	161	642	5506	1256	0	0	156	6	6	1	**1**	D1	2	2	4	4	2	0%	8%	16%	73%	834.5
truss	1000	1000	3564	986	3540	4792	4999	4137	6	1810	6	7	3	286	ID1	2	288	288	288	1000	0%	70%	4%	23%	2347.0
$\sum$	22911	22911	108546	14614	82679	169450	867384	285672	37498	1909629	8022	15550	1455	4423		1026	4841	4987	4962	20893	1%	82%	6%	8%	85517.1

Table 1: Decomposing linear programming basis matrices.

from the size of the branch-and-bound tree and the number of LPs solved. It is true, however, that the lower bound improves fast at first while stalling occurs in later stages of the computation although still large numbers of cuts are found and the problem is finished by branch-and-bound. The same behaviour has been reported for similar problems like the node capacitated graph partitioning problem discussed in Ferreira, Martin, de Souza, Weismantel, and Wolsey [1994].

Investigating the "*primal side*", we see that the greedy heuristic seems to be most reliable. The two dual methods perform exactly the same and yield solutions of the same quality as the greedy. Bin-packing is either very good (a rather rare event) or a catastrophe, but complements the other heuristics. If we look at the quality of the solutions found at the root node as a measure of the method as a stand-alone decomposition heuristic, the table shows pretty good results for the small problems. For the larger instances the situation is a bit different. We detect larger gaps, see, for instance, `scfxm1` or `ship12s`. In fact, we have often observed in longer runs on larger examples that the best solution could steadily be improved and the optimal solution was found late. A reason might be that the heuristics are closely linked to the LP-fixings and essentially always find the same solutions until the branching process forces them strongly into another direction. We believe (and for some we know) that many of the larger problems can be decomposed much better than the *Ub*-column indicates and that there might be potential to further improve stand-alone primal heuristics.

The answer to the final question whether LP basis matrices are decomposable into four blocks is 'Yes' and 'No'. Some like `recipe` or `standata` are (only one row is in the border), others are not, for example `israel`: 98 out of 163 are in the border. The results leave, however, the possibility that larger LP-matrices, that are generally sparser than small ones, can be decomposed better so that we can not give a final answer to this question.

5.2 The `Miplib` Problems

In this test series we examine whether matrices arising from *integer programs* can be decomposed into (bordered) block diagonal form. There are two applications here.

First, decomposing the original constraint matrix A of some general integer program can be useful to *tighten* its LP-relaxations within a branch-and-cut algorithm. The structure of the decomposed matrix is that of a multiple knapsack or general assignment problem, and inequalities known for the associated polytopes (see Gottlieb and Rao [1990], Ferreira, Martin, and Weismantel [1996]) are valid for the MIP under consideration. The first interesting case in this context are two blocks and we set $\beta := 2$. We used $\kappa := \frac{(\#rows)\cdot 1.05}{2}$ rounded up as the block capacity, which allows a deviation of 10% of the actual block sizes in the decomposition.

Table 2 shows the *results* that we obtained for matrices of mixed integer programs taken from the `Miplib`† and preprocessed with the presolver of the general purpose MIP-solver `SIP` that is currently under development at the Konrad-Zuse-Zentrum. We again considered all instances with up to 1,000 rows.

The picture here is a bit different from the one for the linear programming problems. Since the number of blocks is $\beta = 2$ instead of $\beta = 4$, the IP-formulation is much smaller: The number of variables is only one half, the number of conflicting assignments for two adjacent rows is only 4 instead of 12. In addition, there is much less symmetry in the problem. Consequently, the number of cuts does not increase to the same extent, the LPs are smaller and easier to solve (the percentage of LP-time in column *LP* decreases). We can solve instances up to 200 rows and many of the small ones within seconds. Note that $\beta = 2$, on the other hand, leads to doubled block capacities. This means that it becomes much more difficult to separate inequalities that have this number as their right-hand side and have a large or combinatorially restrictive support like z-clique, bin-packing, or composition of clique inequalities, see column *BCC*.

On the *primal side* the results are similar to the `Netlib` instances, but the heuristics seem to perform a little better for two blocks than for four.

How decomposable are the MIP-matrices? We see that not all, but many of the larger problems can be brought into bordered block diagonal form (the small ones can not). Of course, there are also exceptions like the `air`-problems which were expected to be not decomposable. Anyway, there seems to be potential for the multiple-knapsack approach and further research in this direction, especially because there are only very few classes of cutting planes known for general MIPs.

†Available from URL `http://www.caam.rice.edu:80/~bixby/miplib/miplib.html`.

Name	Original			Presolved		Cuts					B&B		Best Solutions				Heuristics at Root				Time				
	rows	col	nz	col	nz	Init	Cov	2part	BCC	Pool	Nod	Iter	Lb	Ub	He	No	G	D1	D2	B	Cm	LP	Sep	Heu	Tot
mod008	6	319	1243	1	6	23	0	0	0	0	1	1	3	**3**	IG	1	3	3	3	6	0%	6%	6%	0%	0.1
stein9	13	9	45	9	45	50	115	64	1	23	16	34	7	**7**	IG	1	7	7	7	13	6%	34%	24%	6%	0.3
p0040	13	40	70	30	60	16	0	0	0	0	1	1	3	**3**	*	1	13	13	13	13	0%	0%	40%	0%	0.1
p0033	15	32	97	11	41	41	23	3	0	1	1	3	4	**4**	D1	1	4	4	4	15	0%	50%	16%	0%	0.1
gt1	15	46	92	46	92	19	262	87	0	61	13	33	6	**6**	D1	1	6	6	6	15	4%	47%	26%	4%	0.4
flugpl	16	16	40	6	21	46	16	0	1	0	1	2	1	**1**	ID2	1	3	3	1	16	0%	25%	0%	0%	0.0
bm23	20	27	478	1	20	79	0	0	0	0	1	1	10	**10**	IG	1	10	10	10	20	0%	22%	0%	11%	0.1
enigma	21	100	289	99	287	40	922	216	0	315	28	83	10	**10**	IG	1	10	10	10	21	7%	41%	28%	14%	2.0
air01	23	771	4215	18	125	88	0	0	0	0	1	1	3	**3**	IG	1	3	3	3	23	0%	7%	0%	0%	0.4
rgn	24	180	460	33	102	64	724	163	1	324	13	47	5	**5**	IG	1	5	5	5	24	12%	48%	18%	10%	1.2
pipex	25	48	192	48	192	46	421	180	0	201	13	33	9	**9**	IG	1	9	9	9	25	12%	42%	23%	7%	0.8
lseu	28	88	308	50	185	68	98	43	0	19	1	5	7	**7**	D1	1	7	7	7	28	8%	30%	43%	4%	0.2
gt2	28	173	346	173	346	39	1333	340	0	290	28	86	11	**11**	IG	1	11	11	11	28	8%	46%	28%	8%	2.8
sentoy	30	60	1800	1	30	119	0	0	0	0	1	1	15	**15**	IG	1	15	15	15	30	0%	18%	12%	0%	0.2
stein15	36	15	120	15	120	142	17654	13145	0	26682	5458	7595	18	**18**	IG	1	18	18	18	36	18%	27%	22%	16%	90.1
misc02	43	55	405	46	368	153	2455	1065	0	1612	34	110	15	**15**	IG	9	17	21	21	43	6%	59%	18%	11%	9.2
sample2	45	64	140	55	131	81	976	171	1	331	7	29	4	**4**	D1	1	4	4	4	45	6%	61%	21%	6%	2.0
air02	50	6774	61555	76	897	270	4	0	34	0	1	3	22	**22**	IG	1	22	24	24	50	0%	6%	18%	0%	6.0
misc01	54	79	729	66	678	218	4556	2762	0	4861	367	581	24	**24**	IG	17	25	26	26	54	7%	42%	29%	14%	31.4
mod013	62	96	192	48	144	254	1810	108	1	722	13	43	6	**6**	D1	1	6	6	6	62	6%	68%	17%	3%	5.1
mod014	74	86	172	43	129	246	148	0	1	9	1	3	2	**2**	D1	1	2	2	2	74	5%	32%	35%	8%	0.3
lp4l	85	1086	4677	791	3462	277	23233	16941	0	45877	1129	1779	35	**35**	IG	487	39	41	41	85	4%	13%	27%	52%	481.2
bell5	87	101	257	73	215	171	2418	473	1	800	13	38	4	**4**	ID1	4	6	7	7	87	5%	67%	15%	9%	10.6
p0291	92	103	373	63	279	365	2943	366	0	757	19	54	7	**7**	D1	1	7	7	7	92	7%	52%	21%	15%	10.7
misc03	96	154	2023	133	1934	360	50431	32949	1	85357	12745	15609	43	44	IG	9	46	46	46	96	5%	11%	39%	38%	1800.0
l152lav	97	1989	9922	695	3712	308	32271	23459	0	71535	1525	2276	36	**36**	IG	1102	43	43	43	97	3%	16%	24%	53%	754.5
khb05250	100	1299	2598	1275	2574	196	12634	1917	1	3421	73	227	25	**25**	IG	1	25	25	25	100	3%	25%	42%	25%	89.0
harp2	100	1373	2598	1225	2450	116	250	158	0	3	1	4	17	**17**	D1	1	17	17	17	100	0%	2%	94%	0%	20.4
bell4	101	114	293	84	248	191	7162	1041	1	2157	49	145	5	**5**	IB	4	11	11	11	101	6%	55%	17%	17%	31.7
bell3a	107	121	311	89	263	203	2718	492	1	802	10	32	4	**4**	B	6	13	13	13	107	5%	63%	16%	12%	14.5
bell3b	107	121	311	89	263	203	2718	492	1	802	10	32	4	**4**	B	6	13	13	13	107	5%	63%	15%	13%	14.6
p0201	113	195	1677	177	1527	200	9230	2390	1	5757	46	128	21	**21**	IG	24	30	30	30	113	3%	64%	12%	19%	125.4
stein27	118	27	378	27	378	353	280101	82635	3	382512	1753	3874	32	56	IG	1	56	56	56	118	12%	44%	34%	6%	1800.1
air03	124	10757	91028	656	5878	830	21108	9064	0	24479	130	377	49	**49**	IG	13	58	59	58	124	1%	57%	27%	12%	1146.7
p0808a	136	240	480	120	304	392	14298	1463	1	4711	49	175	8	**8**	D1	1	8	8	8	136	6%	62%	16%	13%	83.2
mod010	146	2655	11203	1973	8404	453	53084	33036	1	118368	727	1170	47	59	IG	28	61	66	66	146	2%	24%	25%	46%	1802.5
blend2	169	319	1279	88	1039	515	0	0	0	0	4	4	10	**10**	IG	1	10	10	10	169	2%	57%	22%	1%	1.7
noswot	182	127	732	50	455	745	84957	34285	2	533391	1006	1646	10	15	IG	10	37	42	42	182	10%	47%	33%	6%	1800.4
10teams	210	1600	9600	1600	9600	210	67719	25710	1	127827	130	418	43	90	D1	1	90	90	90	210	3%	58%	30%	6%	1804.0
misc07	224	254	8589	229	8474	894	10476	8516	0	13110	118	119	57	93	IG	11	101	107	107	224	0%	23%	22%	52%	1836.5
vpm1	234	378	749	203	574	906	18959	446	1	4060	34	117	7	**7**	IG	1	7	14	14	234	3%	80%	9%	5%	322.7
vpm2	234	378	917	203	574	906	18959	446	1	4060	34	117	7	**7**	IG	1	7	14	14	234	3%	79%	9%	6%	317.4
p0808aCUTS	246	240	839	230	828	612	15096	1731	1	3058	22	84	8	**8**	D1	1	8	8	8	246	4%	65%	12%	17%	231.7
p0548	257	477	1522	250	1009	523	94934	20509	1	94842	175	462	25	49	IG	35	68	68	68	257	5%	69%	12%	12%	1800.1
misc05	266	131	2873	129	2869	581	16094	4604	0	5545	43	80	28	116	IG	13	120	126	126	266	1%	89%	5%	3%	1835.0
modglob	289	420	966	356	902	865	75843	1685	1	29153	148	422	8	**8**	IG	18	12	12	12	289	3%	74%	11%	9%	1512.6
gams	291	556	2431	540	2400	1622	41190	11	0	6897	169	295	15	21	B	12	36	36	36	291	1%	84%	9%	3%	1801.1
fiber	297	1232	2644	418	1254	593	32222	7410	1	19253	31	123	9	22	ID1	3	38	38	38	297	2%	86%	4%	6%	1807.5
p0282	305	202	1428	168	841	609	56425	17488	1	52114	91	252	23	36	D1	1	36	36	36	305	3%	71%	9%	14%	1803.2
stein45	331	45	1034	45	1034	992	25305	11011	0	22779	28	88	11	154	IG	2	157	157	157	331	1%	89%	5%	2%	1820.3
qnet1_o	369	1454	4040	837	2474	699	34759	9401	1	19450	37	111	12	28	D1	1	28	28	28	369	2%	74%	6%	15%	1812.7
qnet1	407	1454	4405	924	2843	653	24614	8345	1	16223	22	69	8	35	D1	1	35	35	35	407	2%	77%	5%	14%	1801.5
air05	408	7195	50762	3104	23458	600	5677	1286	0	207	13	19	19	189	IG	1	189	194	194	408	0%	67%	20%	10%	1868.9
set1ch	477	697	1382	445	1130	1437	36069	2756	1	9871	25	89	7	12	D1	1	12	12	12	477	3%	72%	7%	16%	1804.4
fixnet3	478	878	1756	498	1374	1990	23068	145	1	959	34	83	12	13	D1	1	13	13	13	478	1%	88%	4%	4%	1809.4
fixnet4	478	878	1756	498	1374	1990	26165	192	1	1860	34	97	13	**13**	D1	1	13	13	13	478	1%	85%	5%	6%	1789.9
fast0507	484	63001	406865	4927	31419	659	9549	4335	0	2135	10	24	5	183	IG	3	217	217	217	484	1%	51%	15%	26%	1800.1
set1al	492	712	1412	460	1160	1452	27188	2041	1	5790	22	65	6	12	D1	1	12	12	12	492	2%	78%	5%	13%	1816.9
set1cl	492	712	1412	460	1160	1452	26700	2001	1	5711	22	64	6	12	D1	1	12	12	12	492	2%	77%	5%	14%	1805.3
gen	622	797	2064	360	1303	2076	15337	304	1	3914	13	30	5	19	ID1	1	19	19	19	622	1%	68%	5%	24%	1809.8
mod015	622	797	2064	360	1303	2076	14722	299	1	3398	13	28	5	19	ID1	1	19	19	19	622	1%	68%	4%	24%	1807.0
danoint	664	521	3232	521	3232	1016	13895	5307	1	4119	7	24	4	188	ID1	2	189	189	189	664	1%	77%	6%	14%	1861.4
misc06	696	1572	5126	949	3106	1298	15987	3428	1	4911	13	28	4	70	ID1	1	70	70	70	696	2%	62%	7%	27%	1835.3
air06	763	8572	67571	4092	34800	1145	7622	3531	1	732	4	12	3	354	IG	2	356	363	362	763	1%	43%	35%	18%	1934.3
air04	782	8904	70189	4154	35000	1052	6242	3243	1	434	4	10	3	355	IG	1	355	355	355	782	0%	34%	26%	37%	1913.6
adrud	795	998	15876	495	8479	6342	3955	287	0	209	25	32	10	32	D1	1	32	32	32	795	0%	77%	10%	10%	1835.6
$\sum$	14814	134914	876632	35938	221378	43230	1395844	405976	75	1778801	26610	39627	905	2729		1863	2931	2990	2986	14814	2%	63%	15%	17%	56337.8

Table 2: Decomposing matrices of mixed integer programs.

The second application of matrix decomposition to integer programming is a *new branching rule.* Decomposing the transposed constraint matrix will identify the variables in the border as linking variables that are interesting candidates for branching. Since most MIP-codes create a binary searchtree, we try to decompose these matrices into $\beta := 2$ blocks. As block capacity we use $\kappa := \frac{(\#rows)\cdot 1.05}{2}$ rounded up to obtain two subproblems of roughly the same size. The test set consists of all problems with up to 1,000 rows (1,000 columns in the original problem).

Table 3 shows the *results* of our computations. Surprisingly, the performance of our algorithm is not only similar to the "primal" case, in fact it is even better! We can solve almost all problems with up to 400 rows. One reason for this is that MIPs tend to have sparse columns, but not necessarily sparse rows. Dense columns in the transposed matrices (dense rows in the original ones) leave less freedom for row assignments, there are fewer possibilities for good decompositions, and the LP-relaxations are tighter than in the primal case.

For the reason just mentioned we expected that the transposed problems would be less decomposable than the originals and it came as a surprise to us that the "dual" problems decompose nearly as well as the primal ones. The transposed matrices have on average about 60 rows in the border in comparison to only 40 for the originals. But the percentage of rows in the border (i.e., the sum of the *Ub* column divided by the sum of the *rows* column) is 18.4% (12.8%) in the primal and 20.6% (25.3%) in the dual case (the values in parentheses count only instances that were solved to optimality). Note, however, that the dual values are biased heavily by the (not decomposable) `adrud` instance. An explanation may be that many of these problems contain a few important global variables that migrate into the border.

Name	`SIP` without MAD			`SIP` with MAD			CPLEX		
	gap	nodes	time	gap	nodes	time	gap	nodes	time
bell3a	96.6%	100000	236.0	67.9%	100000	288.2	0.0%	42446	54.9
bell3b	-	100000	142.1	-	100000	120.3	3.8%	100000	120.6
bell4	-	100000	137.4	2.1%	100000	164.5	3.2%	100000	119.0
bell5	6.3%	100000	184.0	2.7%	100000	153.6	6.1%	100000	113.6
noswot	-	100000	219.8	-	100000	195.2	10.3%	100000	228.1

maximum infeasibility branching

bell3a	0.0%	33350	92.1	0.0%	55763	221.4	0.0%	19100	108.9
bell3b	0.0%	25909	308.4	0.0%	25706	301.4	0.0%	6537	70.8
bell4	1.5%	91240	1800.0	2.2%	96628	1800.0	0.0%	47001	710.5
bell5	0.1%	100000	806.4	0.1%	100000	824.0	0.1%	100000	586.9
noswot	27.9%	63474	1800.0	20.9%	90743	1800.0	7.5%	43038	1800.1

strong branching

Table 4: Comparing integer programming branching rules.

We used optimal decompositions of transposed MIP matrices to test our idea to branch on variables in the border first. The computations were performed using the above mentioned MIP-solver `SIP`. As our *test set* we selected all problems that are not extremely easy (less than 10 CPU seconds for `SIP`) and that decompose into bordered block diagonal form with a "relatively small" border: `mod014, bell3a, bell3b, bell4, bell5, noswot, blend2, vpm1, vpm2, set1ch, set1al` and `set1cl`. Unfortunately, and contrary to what we had expected, it turned out that `mod014, blend2, vpm1, vpm2, set1ch, set1al` and `set1cl` have only *continuos* variables in the border that do not qualify for branching variables! All border variables in `noswot` are integer, in the `bell`-examples 1 (2) out of 5 (6). We tried to extend the test set by decomposing only the integer part of all these matrices but it failed, because the integer parts turned out to have block diagonal form, i. e., no variables in the border.

For the remaining four `bell*` and the `noswot` example, we performed the following tests. The MIPs were solved with `SIP` using four *different branching strategies.* Maximum infeasibility branching (i.e., branching on a variable that is closest to 0.5) and *strong branching* (cf. Robert E. Bixby, personal communication) are two standard branching strategies for solving mixed integer programs. The other two branching rules result from extending these two methods by our idea of branching on a variable that belongs to the border first. The limit of the computation was 1,800 CPU seconds or 100,000 branch-and-bound nodes, whatever came first.

Table 4 summarizes our results. The version without MAD (MAD stands for the MAtrix Decomposition)

Name	Original			Presolved		Cuts					B&B		Best Solutions				Heuristics at Root				Time				
	rows	col	nz	col	nz	Init	Cov	2part	BCC	Pool	Nod	Iter	Lb	Ub	He	No	G	D1	D2	B	Cm	LP	Sep	Heu	Tot
stein9	9	13	45	1	9	35	0	0	0	0	1	1	5	**5**	IG	1	5	5	5	9	0%	9%	0%	0%	0.1
stein15	15	36	120	1	15	59	0	0	0	0	1	1	8	**8**	IG	1	8	8	8	15	0%	18%	0%	0%	0.2
flugpl	16	16	40	11	30	36	16	2	1	0	1	2	1	**1**	IG	1	1	1	1	16	0%	8%	8%	0%	0.1
bm23	27	20	478	3	78	109	0	0	0	0	1	1	13	**13**	IG	1	13	13	13	27	0%	25%	5%	5%	0.2
stein27	27	118	378	1	27	107	0	0	0	0	1	1	13	**13**	IG	1	13	13	13	27	0%	23%	4%	0%	0.2
p0033	32	15	97	9	59	120	205	51	0	31	7	16	6	**6**	IG	1	6	6	6	32	9%	46%	16%	5%	0.5
p0040	40	13	70	13	70	100	47609	7530	10	27421	1522	3361	13	**13**	IG	1	13	14	14	40	18%	38%	27%	8%	90.5
stein45	45	331	1034	1	45	179	0	0	0	0	1	1	22	**22**	IG	1	22	22	22	45	2%	27%	8%	0%	0.4
gt1	46	15	92	15	92	46	119471	29195	15	71563	4924	8786	16	**16**	IG	20	18	22	22	46	20%	30%	33%	8%	283.8
pipex	48	25	192	19	96	48	142	76	16	0	4	7	20	**20**	IG	2	21	22	22	48	2%	22%	33%	8%	0.4
misc02	55	43	405	18	229	279	547	517	0	562	25	60	27	**27**	D1	1	27	27	27	55	7%	43%	27%	10%	3.8
sentoy	60	30	1800	1	60	239	0	0	0	0	1	1	29	**29**	IG	1	29	29	29	60	1%	33%	9%	1%	0.5
sample2	64	45	140	45	140	82	2223	638	1	1308	13	45	4	**4**	*	9	6	6	6	64	7%	63%	18%	6%	6.6
misc01	79	54	729	23	359	435	2138	1121	0	1229	28	79	38	**38**	IG	1	38	38	38	79	7%	44%	32%	10%	10.3
mod014	86	74	172	74	172	172	6137	697	1	1927	28	109	8	**8**	ID2	1	8	9	8	86	9%	52%	23%	11%	16.7
lseu	88	28	308	20	198	332	10336	3532	0	8335	76	216	26	**26**	*	71	36	42	39	88	7%	64%	15%	10%	65.4
mod013	96	62	192	62	192	192	448495	45097	11	171613	4711	8604	14	20	B	32	23	23	23	96	14%	25%	47%	9%	1800.2
enigma	100	21	289	12	199	375	3695	1802	0	1575	40	110	44	**44**	IG	12	46	48	48	100	6%	40%	37%	11%	19.4
bell5	101	87	257	87	257	185	5598	969	1	1645	40	108	5	**5**	IG	4	9	9	9	101	7%	56%	21%	12%	21.5
p0291	103	92	373	65	319	261	7980	2306	0	3581	61	170	21	**21**	IG	5	24	29	29	103	4%	70%	11%	12%	78.6
bell4	114	101	293	101	293	204	11829	2005	1	3979	109	277	6	**6**	IG	6	9	9	9	114	6%	58%	17%	15%	64.5
bell3a	121	107	311	107	311	217	4322	942	1	1294	10	46	5	**5**	ID1	1	5	5	5	121	5%	63%	17%	11%	23.8
bell3b	121	107	311	107	311	217	4322	942	1	1294	10	46	5	**5**	ID1	1	5	5	5	121	5%	64%	17%	10%	23.7
noswot	127	182	732	103	531	427	55038	4869	1	12790	778	1605	15	**15**	IG	2	19	19	19	127	6%	40%	21%	28%	337.5
misc05	131	266	2873	79	535	289	255523	30000	0	124752	2236	3462	35	49	IG	64	53	58	58	131	8%	38%	30%	19%	1800.0
misc03	154	96	2023	39	769	840	3666	4828	0	8008	328	565	73	**73**	IG	3	74	74	74	154	3%	28%	38%	19%	159.2
gt2	173	28	346	28	346	173	173371	81355	37	280323	1807	2468	57	67	IG	158	76	83	83	173	7%	56%	23%	10%	1800.2
rgn	180	24	460	24	460	730	38963	6126	1	22421	112	347	20	**20**	IG	1	20	78	42	180	5%	67%	14%	11%	474.6
p0201	195	113	1677	77	699	579	181201	53287	74	223647	625	1295	42	76	IG	192	86	93	93	195	8%	54%	26%	8%	1800.0
p0282	202	305	1428	189	1196	407	4311	1449	0	479	13	36	20	**20**	ID1	1	20	20	20	202	2%	74%	9%	13%	122.7
p0808a	240	136	480	136	480	480	106598	14308	1	70632	151	507	11	18	IG	12	32	36	32	240	5%	76%	12%	5%	1802.1
p0808aCUTS	240	246	839	191	729	674	107094	7435	1	58191	202	525	12	17	ID2	28	36	36	36	240	5%	69%	12%	11%	1800.7
misc07	254	224	8589	53	1367	1456	11803	14984	0	28234	757	1117	119	**119**	IG	1	119	121	121	254	2%	18%	40%	29%	939.8
blend2	319	169	1279	160	478	955	103558	8951	1	21986	226	680	38	**38**	D1	1	38	38	38	319	8%	33%	32%	22%	753.6
vpm1	378	234	749	234	749	784	37116	4150	1	14141	37	136	7	**7**	IG	8	51	61	61	378	3%	80%	7%	7%	1402.3
vpm2	378	234	917	66	581	1072	25484	2483	1	6355	19	87	7	**7**	IG	3	21	69	69	378	3%	76%	8%	11%	982.0
modglob	420	289	966	289	966	1014	42137	6770	1	19025	40	116	9	29	IG	12	39	56	39	420	3%	79%	7%	8%	1807.3
p0548	477	257	1522	153	823	1525	36798	3048	1	8883	46	96	29	49	IG	10	79	116	116	477	3%	48%	9%	39%	1803.7
danoint	521	664	3232	544	2336	649	21785	10004	1	12677	13	47	6	210	IG	4	219	238	219	521	2%	79%	8%	9%	1808.8
gams	556	291	2431	91	1096	4003	14962	2869	0	4200	19	35	47	170	IG	1	170	170	170	556	2%	65%	12%	19%	1800.8
set1ch	697	477	1382	477	1382	1657	20852	1581	1	2680	10	34	4	33	IG	4	42	91	91	697	2%	68%	6%	22%	1843.7
set1al	712	492	1412	492	1412	1672	16354	1713	1	2886	7	26	3	40	IG	2	64	101	101	712	2%	76%	5%	15%	1813.6
set1cl	712	492	1412	492	1412	1672	16354	1713	1	2886	7	26	3	40	IG	2	64	101	101	712	2%	76%	5%	15%	1800.2
air01	771	23	4215	22	4185	7186	0	0	0	0	4	2	184	367	IG	1	367	367	367	771	0%	95%	1%	1%	1893.0
gen	797	622	2064	298	1416	2113	16984	1847	1	2813	10	26	4	70	ID2	1	70	84	70	797	2%	70%	7%	19%	1821.7
mod015	797	622	2064	298	1416	2113	16984	1847	1	2813	10	26	4	70	ID2	1	70	84	70	797	2%	70%	7%	19%	1853.2
fixnet3	878	478	1756	478	1756	1638	13145	3715	1	1757	7	18	3	190	IG	1	190	221	209	878	2%	72%	6%	17%	1850.5
fixnet4	878	478	1756	478	1756	1638	13130	3611	1	2272	7	18	3	197	IG	2	209	221	209	878	2%	77%	6%	13%	1873.8
fixnet6	878	478	1756	478	1756	1638	10505	2989	1	901	7	15	3	186	IG	2	209	221	209	878	1%	80%	5%	12%	2087.3
adrud	998	795	15876	1	998	3991	0	0	0	0	1	1	475	**475**	IG	1	475	475	475	998	0%	75%	17%	0%	750.9
$\sum$	14556	10168	72362	6766	35191	45404	2018781	373354	188	1233109	19094	35364	1582	3007		693	3297	3737	3593	14556	4%	66%	13%	14%	41494.7

Table 3: Decomposing transposed matrices of mixed integer programs.

Name	Original			Presolved		Cuts					B&B		Best Solutions				Heuristics at Root				Time				
	rows	col	nz	col	nz	Init	Cov	2part	BCC	Pool	Nod	Iter	Lb	Ub	He	No	G	D1	D2	B	Cm	LP	Sep	Heu	Tot
g353	81	48	276	48	276	336	19768	6311	240	32467	121	385	16	**16**	IG	4	23	23	23	81	5%	77%	11%	3%	180.7
g444	114	39	253	37	251	756	11324	4251	156	22828	56	195	13	**13**	B	36	22	22	22	114	3%	80%	11%	3%	186.9
d677	324	183	984	174	975	3533	13005	7370	56	54768	25	59	7	79	IG	2	98	98	98	324	0%	97%	1%	0%	11168.4
d688	383	230	1350	223	1341	4506	10719	6871	37	32015	19	40	7	108	IG	3	134	140	134	383	0%	97%	1%	0%	10901.3
$\sum$	902	500	2863	482	2843	9131	54816	24803	489	142078	221	679	43	216		45	277	283	277	902	0%	97%	1%	0%	22437.2

Table 5: Decomposing Steiner-tree packing problems.

Name	Original			Presolved		Cuts					B&B		Best Solutions				Heuristics at Root				Time				
	rows	col	nz	col	nz	Init	Cov	2part	BCC	Pool	Nod	Iter	Lb	Ub	He	No	G	D1	D2	B	Cm	LP	Sep	Heu	Tot
m22	9	6	21	6	21	30	37	19	0	12	16	31	5	**5**	G	1	5	9	9	5	6%	46%	20%	6%	0.1
m25	9	6	23	5	22	44	6	20	0	18	13	24	7	**7**	G	2	9	9	9	9	16%	33%	8%	8%	0.1
m33	14	9	40	8	37	38	210	90	0	68	13	33	6	**6**	G	4	8	10	10	8	8%	40%	17%	20%	0.3
m34	14	9	41	9	41	53	181	111	0	69	22	53	8	**8**	G	1	8	10	10	8	23%	28%	21%	4%	0.5
m41	18	9	43	9	43	41	654	241	1	253	28	76	8	**8**	G	1	8	14	14	10	10%	57%	12%	6%	0.9
m44	18	9	55	8	51	89	333	122	0	133	25	56	10	**10**	G	1	10	10	10	10	11%	35%	35%	5%	0.6
m51	21	14	61	12	58	52	977	349	1	441	52	143	11	**11**	G	11	15	15	15	15	9%	53%	18%	8%	2.1
m54	21	14	90	12	83	93	424	211	0	278	34	77	13	**13**	G	6	15	15	15	15	8%	40%	23%	17%	1.3
m61	21	17	69	15	65	52	742	204	0	245	28	73	9	**9**	G	4	11	11	11	11	14%	49%	22%	4%	1.3
m64	21	17	108	15	100	106	417	285	0	376	40	89	15	**15**	G	5	17	17	17	17	10%	33%	23%	22%	1.6
m71	28	15	71	14	69	64	2353	713	0	1171	76	196	12	**12**	G	5	16	16	16	16	13%	54%	17%	7%	5.0
m74	28	15	128	14	122	135	823	575	0	714	88	171	20	**20**	G	1	20	22	22	20	8%	33%	14%	35%	4.2
m81	28	21	86	20	85	74	3169	979	0	1717	268	476	14	**14**	IG	3	16	16	16	16	12%	47%	18%	10%	7.7
m84	28	21	177	19	167	172	475	392	617	347	49	116	22	**22**	*	18	28	28	28	28	7%	28%	23%	35%	3.4
$\sum$	278	182	1013	166	964	1043	10801	4311	619	5842	752	1614	160	160		63	186	202	202	188	11%	43%	19%	16%	29.2

Table 6: Equipartitioning matrices.

uses the original SIP-code and the original branching rules, the MAD-version branches on a variable from the border first. For comparison, we also list the results that can be obtained using CPLEX. The column labeled *gap* reports the duality gap on termination (- means that no feasible solution was found).

The *results* are mixed. When *strong branching* is used CPLEX is overall best and SIP with MAD is basically always worst (note that for example *noswot* the value of the LP-relaxation already provides the value of the optimal integer solution and so the main difficulty here is to find a good feasible solution). If we branch on a most infeasible variable the situation changes a bit in favour of SIP with MAD. In fact, there are two examples where this strategy performs best in terms of the *gap*. Our limited tests can, of course, not provide a definite evaluation of our idea to branch on border variables. But we think that the results show that this idea might have the potential to become a good branching strategy for certain mixed integer programming problems.

5.3 The Steiner-Tree Packing Problems

We also tested some problem instances for which we know in advance that they have bordered block diagonal from. The problems are integer programming formulations for Steiner-tree packing problems, a problem where in some given graph edge sets (so-called Steiner-trees), each spanning some given subset of the node set, have to be simultaneously packed in the graph under capacity restrictions on the edges, see Grötschel, Martin, and Weismantel [1996]. Unfortunately, our branch-and-cut algorithm performs very bad on these examples, although we extended the time limit to 10,800 CPU seconds, see Table 5. One reason for that might be that the rows that are supposed to be in the border have less nonzero entries than those that are expected to be in the blocks. The heuristics and the LP solutions, however, tend to put the rows into the border that have the most nonzeros entries. The “natural decompositions” result in borders of sizes 22, 24, 71, and 82 for problems g353, g444, d677, and d688, respectively.

5.4 The Equipartition Problems

Our last test set consists of equipartition problems introduced by Nicoloso and Nobili [1992]. These have been generated randomly prescribing a certain matrix-density. We have modified our code to handle the additional equipartition constraint. The results are given in Table 6: We can solve all problems within 10 CPU seconds and, as was already known from Nicoloso and Nobili [1992], random matrices of this type do not decompose well.

Summary and Conclusions

We have shown in this paper that it is possible to decompose typical linear and integer programming matrices up to 200 and more rows to proven optimality using a cutting plane approach based on polyhedral investigations of the matrix decomposition problem. It turned out that a substantial number, but not all, LPs decompose well into four blocks, while even many MIPs, as well as some of their transposes, can be brought into bordered block diagonal form with two blocks. We think that these results show a significant potential for methods that can exploit this structure to solve general MIPs. Our decomposition heuristics work well for small instances, but there is room for improvement for problems of large scale, in particular, if more than two blocks are considered.

Acknowledgements

We are grateful to the Fundação Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) and the German Academic Exchange Service (DAAD) for supporting this work. We enjoyed the cooperation a lot and intend to continue. We want to thank Robert E. Bixby and CPLEX Optimization, Inc., who made the newest β-versions of CPLEX and the CPLEX Callable Library available to us, and Paolo Nobili for sending a collection of his equipartition problems.

References

Carøe, C. C. and Schultz, R. (1996). Dual decomposition in stochastic integer programming. Preprint SC 96-46, Konrad Zuse Zentrum für Informationstechnik Berlin. Available at URL http://www.zib.de/ZIBbib/Publications. Submitted to OR Letters.

CPLEX (1997). *Using the CPLEX Callable Library.* ILOG CPLEX Division, 889 Alder Avenue, Suite 200, Incline Village, NV 89451, USA. Information available at URL http://www.cplex.com.

Crama, Y. and Oosten, M. (1996). Models for machine-part grouping in cellular manufacturing. *Int. J. Prod. Res.*, 34(6):1693–1713.

Dentcheva, D., Gollmer, R., Möller, A., Römisch, W., and Schultz, R. (1997). Solving the unit commitment problem in power generation by primal and dual methods. In Bendsøe, M. and Sørensen, M., editors, *Proc. of the 9th Conf. of the Europ. Consortium for Math. in Industry (ECMI), Copenhagen, 1996.* Teubner Verlag, Stuttgart.

Duff, I., Erisman, A., and Reid, J. (1986). *Direct Methods for Sparse Matrices.* Oxford Univ. Press.

Ehrgott, M. (1992). Optimierungsprobleme in Graphen unter Kardinalitätsrestriktionen. Master's thesis, Univ. Kaiserslautern, Dept. of Mathematics.

Ferreira, C. E., Martin, A., de Souza, C. C., Weismantel, R., and Wolsey, L. A. (1994). The node capacitated graph partitioning problem: a computational study. Preprint SC 94–17, Konrad-Zuse-Zentrum für Informationstechnik Berlin. Available at URL http://www.zib.de/ZIBbib/Publications. To appear in *Mathematical Programming.*

Ferreira, C. E., Martin, A., and Weismantel, R. (1996). Solving multiple knapsack problems by cutting planes. *SIAM J. on Opt.*, 6:858 – 877.

Fiduccia, C. and Mattheyses, R. (1982). A linear-time heuristic for improving network partitions. *Proc. 19th. DAC*, pages 175–181.

Gallivan, K., Heath, M. T., Ng, E., Ortega, J. M., Peyton, B. W., Plemmons, R., Romine, C. H., Sameh, A., and Voigt, R. G. (1990). *Parallel Algorithms for Matrix Computations.* SIAM.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, New York.

Gottlieb, E. and Rao, M. (1990). The generalized assignment problem: Valid inequalities and facets. *Math. Prog.*, 46:31–52.

Grötschel, M., Lovász, L., and Schrijver, A. (1988). *Geometric Algorithms and Combinatorial Optimization.* Springer Verlag, Berlin.

Grötschel, M., Martin, A., and Weismantel, R. (1996). Packing steiner trees: A cutting plane algorithm and computational results. *Math. Prog.*, 72:125 – 145.

Gupta, A. (1996). Fast and effective algorithms for graph partitioning and sparse matrix ordering. Technical Report RC 20496, IBM T. J. Watson Res. Center, Yorktown Heights.

Helmberg, C., Mohar, B., Poljak, S., and Rendl, F. (1995). A spectral approach to bandwidth and separator problems in graphs. *Linear and Multilinear Algebra*, 39:73–90.

Hoffman, K. L. and Padberg, M. W. (1993). Solving airline crew-scheduling problems by branch-and-cut. *Mgmt. Sci.*, 39:657–682.

Kumar, V., Grama, A., Gupta, A., and Karypis, G. (1994). *Introduction to Parallel Computing.* The Benjamin/Cummings Publishing Company, Inc.

Lengauer, T. (1990). *Combinatorial Algorithms for Integrated Circuit Layout.* B.G. Teubner, Stuttgart, and John Wiley & Sons, Chichester.

Löbel, A. (1997). *Optimal Vehicle Scheduling in Public Transit.* PhD thesis, Tech. Univ. of Berlin.

Nicoloso, S. and Nobili, P. (1992). A set covering formulation of the matrix equipartition problem. In Kall, P., editor, *System Modelling and Optimization, Proc. of the 15th IFIP Conf., Zürich, Sept. 1991*, pages 189–198. Springer Verlag, Berlin, Heidelberg, New York.

Nobili, P. and Sassano, A. (1989). Facets and lifting procedures for the set covering polytope. *Math. Prog.*, 45:111–137.

Padberg, M. W. (1973). On the facial structure of set packing polyhedra. *Math. Prog.*, 5:199–215.

Pothen, A., Simon, H. D., and Liou, K.-P. (1990). Partitioning sparse matrices with eigenvectors of graphs. *SIAM J. on Matrix Analysis*, 11(3):430–452.

Rothberg, E. and Hendrickson, B. (1996). Sparse matrix ordering methods for interior point linear programming. Technical Report SAND96-0475J, Sandia National Laboratories.

Schrijver, A. (1986). *Theory of Linear and Integer Programming.* John Wiley & Sons, Chichester.

Sheble, G. and Fahd, G. (1994). Unit commitment literature synopsis. *IEEE Transactions on Power Systems*, 9:128–135.