SOLVING GRAPH BISECTION PROBLEMS WITH SEMIDEFINITE PROGRAMMING

STEFAN E. KARISCH, FRANZ RENDL, AND JENS CLAUSEN

Abstract. An exact solution method for the graph bisection problem is presented. We describe a branch-and-bound algorithm which is based on a cutting plane approach combining semidefinite programming and polyhedral relaxations. We report on extensive numerical experiments which were performed for various classes of graphs. The results indicate that the present approach solves general problem instances with $80 - 90$ vertices exactly in reasonable time, and provides tight approximations for larger instances. Our approach is particularly well suited for special classes of graphs as planar graphs and graphs based on grid structures.

1. Introduction

We consider the problem of partitioning the vertices of a graph into two components. Given is an undirected edge-weighted graph $G(V,E)$, where V denotes the vertex set consisting of n vertices, and E the edge set. The weight of the edges are given by the *Laplace matrix* L, which is defined through the *adjacency matrix* A of the graph by $L := \mathrm{Diag}(Ae_n) - A$. Here, e_n is the n-vector of all ones and the linear operator $\mathrm{Diag}(\cdot)$ forms a diagonal matrix of the row sums Ae_n of A. The *graph bisection problem* is to partition the vertex set V into two components V_1 and V_2 of prespecified sizes $n_1 = |V_1|$ and $n_2 = |V_2|$ $(n = n_1 + n_2)$, such that the total weight of the edges having one vertex in V_1 and one in V_2 is minimized. These edges are said to be *cut* by the partition. We assume that $n_1 \geq n_2$ and denote by $d := n_1 - n_2 \geq 0$ the difference between the number of vertices belonging to the two components of the partition.

We introduce a $(-1,1)$-vector x of length n and represent each vertex by an entry of x. The interpretation of x is that elements of the same sign correspond to vertices belonging to the same component of the partition. Then, the graph bisection problem can be written as

$$(BIS) \qquad z^* := \min\{\tfrac{1}{4}x^tLx : x \in \mathcal{F}\} \tag{1}$$

where

$$\mathcal{F} := \{x : x^te_n = d,\ x \in \{-1,1\}^n\} \tag{2}$$

is the set of feasible solutions or *bisections*. Allowing d to be negative would result in a symmetric solution where x is replaced by $(-x)$. The special case $d = 0$ is called the *equicut problem*.

Key words and phrases. Graph bisection, semidefinite programming.

The first author was supported by the Austrian "Fond zur Förderung der wissenschaftlichen Forschung, Erwin-Schrödiger-Auslandsstipendium Nr. J01204-MAT" while staying at the Univeristy of Copenhagen from April 1996 until March 1997.

The graph bisection problem has applications in various fields. It is used to model for example problems arising in VLSI design [3], scientific computing [35], sparse matrix computation [2], physics [3] or parallel programming [10]. Also, the partitioning of graphs into several components is often done by recursive bisections, see e.g. [14]. For a survey on the application in the context of layout problems we refer to [30].

The graph bisection problem is known to be NP-hard. An exact solution method for the equicut problem based on branch-and-cut and linear programming relaxations was proposed by Brunetta, Conforti and Rinaldi [5]. For the more general node capacitated graph partition problem, which is also referred to as min-cut clustering problem, Ferreira et al. [14] described a similar approach, while Johnson, Mehrotra and Nemhauser [24] suggested an approach based on column generation. A parallel solution method for graph partition problems based on simple bounding functions was given by Clausen and Träff [7]. All these exact solution methods have been limited to instances of general graphs with around 60 vertices.

Many heuristic methods have been proposed in the literature, see e.g. [6, 10, 23, 28, 31, 34], which are designed to approximate large scale problems as they appear in real world applications. Nevertheless, it is a great challenge to develop exact methods for general problem graphs with more than 100 vertices. The approach presented in this article is certainly a step in this direction.

Semidefinite programming has been successfully applied to various graph optimization problems. Goemans and Williamson [16] presented in their seminal work an .878-approximation algorithm for the *max-cut* problem, using a certain semidefinite relaxation of the problem. Subsequent work of Frieze and Jerrum [15] provided theoretical approximations for *max-k-cut* and *max-bisection problems*. For the latter problem they obtained a polynomial approximation algorithm with a performance guarantee of .65.

From a practical side, Helmberg [17] and Helmberg et al. [20] applied with great success semidefinite programming in combination with polyhedral relaxations to max-cut. Subsequent work extended the range of the dimensions of exactly solvable problem instances considerably, see [19]. Karisch and Rendl [26] investigated semidefinite programming relaxations of graph equipartition problems. Their results were based on earlier work of Alizadeh [1] and provided tight relaxations for graph equipartitioning. Wolkowicz and Zhao [36] derived semidefinite relaxations for general graph partition problems. For recent investigations on the potentials and limitations of cutting plane algorithms for semidefinite relaxations we refer to [22].

In this work, we derive a branch-and-bound method for the general graph bisection problem. It is based on a cutting plane approach combining semidefinite and polyhedral relaxations. Our goal is to keep the presentation rather selfcontained. Therefore we also include details, which are well known to readers familiar with recent work in this field. The practical performance of our method shows that semidefinite programming enables exact solution of general problem instances with $80-90$ vertices and tight approximation of bisections of larger graphs. Our approach is particularly well suited for special classes of graphs as planar graphs and graphs based on grid structures.

The paper is organized as follows. We conclude this section by introducing basic notation which will be used throughout the paper. Section 2 reviews lower bounds for the graph bisection problem which are based on nonlinear relaxation and addresses how these can be tightened employing polyhedral information. We derive the branch-and-bound algorithm in Section 3 and describe algorithmical aspects in Section 4. Section 5 contains our numerical experiments and results. Finally, we give some conclusions and directions to future work.

1.1. **Basic Notation.** The following notation will be used in the remainder of the paper.

We work in the space of real $n \times k$ matrices $\Re^{n\times k}$. This space is considered with the *trace inner product* tr A^tB for $A, B \in \Re^{n\times k}$. The *identity matrix* in $\Re^{n\times n}$ is denoted by I_n, the matrix $E_n \in \Re^{n\times n}$ is the *matrix of all ones.* The *n-vector of all ones* is written as e_n. Subscripts will not be used if the dimensions are clear from the context. The *j-th canonical unit vector* is written as u_j. By $\cdot^\perp$ we denote the *orthogonal complement*, e.g. $e^\perp$ stands for the orthogonal complement of the vector of all ones. The linear operator $\mathrm{Diag}(d)$ forms a diagonal matrix from a vector $d \in \Re^n$. Its *adjoint operator* $\mathrm{diag}(D)$, acting on $D \in \Re^{n\times n}$, yields a vector containing the diagonal elements of D.

The vector containing the eigenvalues of a symmetric matrix $A \in \Re^{n\times n}$ is denoted by $\mu(A)$, and the *minimal eigenvalue* is given by $\mu_{\min}(A)$. For symmetric matrices A and B, the *Löwner partial order* is $A \succeq B$ ($A \succ B$), meaning that $A-B$ is positive semidefinite (positive definite). The *Hadamard product* of two matrices $A, B \in \Re^{n\times k}$ is defined as $A \circ B = (a_{ij}) \circ (b_{ij}) := (a_{ij} \cdot b_{ij})$.

2. Lower Bounds based on Nonlinear Relaxation

2.1. **Semidefinite Relaxation.** In this section we review eigenvalue bounds and bounds based on semidefinite relaxation for the graph bisection problem. We also discuss their relationship and derive the semidefinite relaxations used for the bounding in the branch-and-bound algorithm.

The first lower bounds based on orthogonal relaxation were introduced by Donath and Hoffmann [11] in the early 1970's. Instead of the $(-1,1)$-model these relaxations base on a $(0,1)$-model. Here, an $n \times 2$ $(0,1)$-matrix Y is used whose rows correspond to the vertices of the graph and whose columns represent the two components in which V is to be partitioned. An entry y_{ij} of this matrix is 1 if vertex i belongs to component V_j in the partition. Now an equivalent formulation of (BIS) is

$$(3)\quad (BIS) \qquad z^* = \min\{\tfrac{1}{2}\mathrm{tr}\ Y^tLY : Ye_2 = e_n,\ Y^te_n = (n_1,n_2)^t,\ Y \in \{0,1\}^{n\times 2}\}.$$

Donath and Hoffmann [11] used the fact that the columns of a feasible Y in the $(0,1)$-model are orthogonal. Thus, the feasible set can be relaxed to all $Y \in \Re^{n\times 2}$ that satisfy

$$Y^tY = \begin{bmatrix} n_1 & 0 \\ 0 & n_2 \end{bmatrix}.$$

Boppana [4] tightened this relaxation of the graph bisection problem by including the constraints on the row and column sums of Y. Independently, Rendl and Wolkowicz [33] proposed the same idea for the general graph partition problem, which contains (BIS) as special case. These bounds are obtained by projecting on the linear manifold

$$\{Y : Ye_2 = e_n,\ Y^te_n = (n_1,n_2)^t,\ Y \in \Re^{n\times 2}\}$$

and solving

$$(BIS_{BRW}) \qquad \max_{f^t e=0} \tfrac{n^2-d^2}{4n}\mu_{\min}(V^t(L+\mathrm{Diag}(f))V) \tag{4}$$

where V contains an orthonormal basis of $e^\perp$, i.e $V^te = 0$, $V^tV = I_{n-1}$, and f is a vector.

Alizadeh [1] observed first that the eigenvalue bounds can be viewed as dual problems to semidefinite relaxations of graph partition problems. Poljak and Rendl [32] derived semidefinite relaxations for the graph bisection problem yielding the same bound as (BIS_{BRW}). For general graph equipartition problems these equivalences were obtained by Karisch and Rendl [26].

We are now going to relax the $(-1,1)$-model. As observed in many places we just have to linearize the objective function by introducing a new variable $X := xx^t$. An entry x_{ij} of X equals $+1$ if i and j are in the same set, and -1 otherwise. The set of feasible bisections can be also expressed using X and is given by

$$\begin{aligned}\mathcal{T} &:= \{X : X = xx^t, x \in \mathcal{F}\}\\ &= \{X : \mathrm{diag}(X) = e_n, e_n^tXe_n = d^2, \mathrm{rank}(X) = 1, X \succeq 0\}.\end{aligned} \tag{5}$$

Thus, we can restate (BIS) as

$$(BIS) \qquad z^* = \min\{\tfrac{1}{4}\mathrm{tr}\ LX : X \in \mathcal{T}\}. \tag{6}$$

Dropping the rank condition on X yields a semidefinite relaxation of (BIS)

$$(BIS_{SDP}) \qquad \min\{\tfrac{1}{4}\mathrm{tr}\ LX : \mathrm{diag}(X) = e_n,\ e_n^tXe_n = d^2,\ X \succeq 0\} \tag{7}$$

which provides the same bound as (BIS_{BRW}), see [32].

2.2. **Tightening the Relaxation with Polyhedral Information.** As shown in different places, e.g. [18, 19, 26], combining semidefinite relaxations with polyhedral information provides very tight relaxations. The goal is to find tighter descriptions of $\mathcal{T}$, or to be more precise of its convex hull, than suggested by the feasible set of (BIS_{SDP}).

Generic inequalities for $(-1,1)$-problems are the so called *hypermetric inequalities*. They base on the fact that for any $x \in \{-1,1\}^n$, the inequality $|h^tx| \geq 1$ is valid, if $h \in \Re^n$ is integer and h^te is odd. With respect to the semidefinite relaxations, these inequalities are just

$$|h^tx| \geq 1 \iff h^txx^th \geq 1 \iff \mathrm{tr}\ (hh^t)X \geq 1.$$

The simplest hypermetric inequalities are the *triangle inequalities*, where h contains all zeros except three elements which are either $+1$ or -1. They read,

$$\begin{array}{rl} x_{ij} + x_{ik} + x_{jk} \geq -1 & (h_i = +1, h_j = +1, h_k = +1)\\ x_{ij} - x_{ik} - x_{jk} \geq -1 & (h_i = +1, h_j = +1, h_k = -1)\\ -x_{ij} + x_{ik} - x_{jk} \geq -1 & (h_i = +1, h_j = -1, h_k = +1)\\ -x_{ij} - x_{ik} + x_{jk} \geq -1 & (h_i = -1, h_j = +1, h_k = +1)\end{array} \tag{8}$$

for distinct triples of vertices (i,j,k), where we use the fact that X is symmetric and has diagonal elements equal to 1. We are only going to use the triangle inequalities here, since they provide a sufficiently tight relaxation when they are added. For further details and polyhedral results regarding the special case of equicut, we refer to [5, 8, 9].

The number of possibly violated inequalities is $\mathcal{O}(n^3)$. We will describe the triangle inequalities in the resulting tighter relaxation in compact form. Suppose there are m_b inequalities in a particular relaxation. We introduce a linear operator $\mathcal{B} : \Re^{n\times n} \to \Re^{m_b}$ acting on X, and a vector $b \in \Re^{m_b}$, which both take care of the inequalities in the model. The r-th inequality is given by

$$\mathcal{B}_r(X) + b_r = \text{tr } h_{(r)}h_{(r)}^t X + b_r \geq 0,$$

with the vector $h_{(r)} \in \Re^n$ defining a triangle inequality as in (8), and $b_r = -1$.

The (tighter) semidefinite relaxation including (some) triangle inequalities is written as

$$\text{(9)} \qquad (P) \qquad \min\{\tfrac{1}{4}\text{tr } LX : \text{diag}(X) = e_n,\ e^tXe = d^2,\ \mathcal{B}(X) + b \geq 0,\ X \succeq 0\}.$$

The cutting plane approach based on this relaxation will provide the lower bounds for the branch-and-bound algorithm described in the following section.

Before going on, we have to address a difficulty that arises in solving semidefinite relaxations with equality constraints. Since we are going to apply an interior point method to solve the semidefinite programming problem (P), we have to make sure that there are *strictly feasible solutions*, i.e. feasible solutions X for which $X \succ 0$ and $\mathcal{B}(X) + b > 0$ holds. It is obvious that there exists no positive definite solution in the equicut case, i.e. if $d = 0$, since the constraint $e^tXe = 0$ makes the smallest eigenvalue of X equal to 0. We are going to show in Section 4.1 how one can transform an equicut problem into an equivalent one, such that the new problem has strictly feasible points. Nevertheless, we also have to ensure that the feasible set of (P) with $d > 0$ has interior points. This becomes of importance when several variables are fixed by branching. In Section 3.2 we are going to identify situations in which subproblems of (P) lack interior, even if $d > 0$.

3. The Branch-and-Bound Algorithm

In this section we describe a branch-and-bound algorithm for the graph bisection problem in detail. Even though the basic ideas are straight forward, we have to consider modifications in the relaxations for bounding certain subproblems due to the above described difficulties regarding interior points.

3.1. **Branching.** The branching is based upon the decision whether two vertices belong to the same set or not, which results in a binary branching tree. In order to reduce the dimension of the subproblems in lower levels of the tree we "merge" vertices when we branch. We describe the branching in the root node of the search tree in a general way, where we allow any pair of vertices to be chosen. Then we will restrict this choice to be able to manage consecutive branchings. We assume that the larger component of the partition is always the first one, i.e. the corresponding entries of x are $+1$.

In the following we use a superscript on the problem to indicate the level of the search tree. Hence the original problem in the root node is

$$\text{(10)} \qquad (BIS^0) \qquad z^0 = \min\{\tfrac{1}{4}x^tL^0x : x^tw^0 = d,\ x \in \{-1,1\}^n\}$$

where we define $w^0 := e$ as the *weight vector* of the problem.

We now describe how the branching and generation of the two subproblems in the first level of the branching tree are carried out. Suppose that p and q are the vertices on which the decision in the branching is made and let $p < q$.

The first possibility is that they go into the same set, which is equivalent to adding the extra constraint $x_p = x_q$ in (BIS^0). The bisection problem with this extra constraint is equivalent to the following lower dimensional problem. Let $L^1 = (l^1_{ij})$ be an $(n-1)\times(n-1)$ matrix obtained from L^0 by replacing the p-th row and column by the sum of rows and columns p and q, respectively, and deleting the q-th row and column. For the sum constraints we introduce the $(n-1)$-dimensional weight vector w^1 such that

$$w^1_i := \begin{cases} w^0_i + w^0_q & \text{if } i = p \\ w^0_i & \text{otherwise} \end{cases} \tag{11}$$

This weight vector counts the merged vertices twice in the sum constraints. Now, (BIS^0) as given in (10) with the extra constraint $x_p = x_q$ is equivalent to

$$(BIS^1) \qquad \min\{\tfrac{1}{4}x^tL^1x : x^tw^1 = d,\ x \in \{-1,1\}^{n-1}\}.$$

Thus, the semidefinite relaxation of (BIS^1) is

(12)

$$(P^1) \qquad \min\{\tfrac{1}{4}\mathrm{tr}\ L^1X : \mathrm{diag}(X) = e_{n-1},\ (w^1)^tXw^1 = d^2,\ \mathcal{B}(X) + b \geq 0,\ X \succeq 0\}.$$

where the matrix variable is an $(n-1)\times(n-1)$ matrix. As inequality constraints we now consider the generic triangle inequalities for an $(n-1)$-dimensional problem.

In the case, where the vertices p and q are separated, i.e. $x_p = -x_q$, we perform a "switching" in the problem first, before we reduce it to a smaller dimensional one. A switching is obtained in the following way. Define the vector $y \in \Re^n$ with

$$y_i := \begin{cases} -1 & \text{if } i = q \\ 1 & \text{otherwise} \end{cases}$$

and $\bar{x} := \mathrm{Diag}(y)x$ where x is the original n-dimensional $(-1,1)$-variable of (BIS^0). Then $\bar{x} \in \{-1,1\}^n$, and we have $x^tL^0x = \bar{x}^t\bar{L}\bar{x}$ with $\bar{L} := \mathrm{Diag}(y)L^0\mathrm{Diag}(y)$. For the sum constraint we have $x^tw^0 = x^t\mathrm{Diag}(y)\mathrm{Diag}(y)w^0 = \bar{x}^t\bar{w}$ where $\bar{w} := \mathrm{Diag}(y)w^0$. Since $x_p = -x_q$ if and only if $\bar{x}_p = \bar{x}_q$, (BIS^0) from (10) with the extra constraint $x_p = -x_q$ is equivalent to the *switched problem*

$$\min\{\tfrac{1}{4}\bar{x}^t\bar{L}\bar{x} : \bar{x}^t\bar{w} = d,\ \bar{x}_p = \bar{x}_q,\ \bar{x} \in \{-1,1\}^n\}.$$

Now we can proceed as we did above by merging p and q in the switched problem. To obtain the new cost matrix L^1 from L^0, we replace the p-th row and column by the difference of rows and columns p and q, respectively, and delete the q-th row and column which results in an analogous update as before. The new weight vector $w^1 \in \Re^{n-1}$ is

$$w^1_i := \begin{cases} w^0_i - w^0_q & \text{if } i = p \\ w^0_i & \text{otherwise} \end{cases} \tag{13}$$

Note that when switched first, the first entry of the weight vector w^1 after the merging becomes 0. This means that the signs of the remaining (unassigned) vertices must satisfy the cardinality constraint.

To distinguish between the two ways of generating subproblems we refer to the first as a *pure merging* and to the second as a *switched merging*.

In order to make consecutive mergings manageable we restrict the choice for the vertex pair to branch on and assume that vertex 1 will always play the role of p. This means that vertex 1 becomes a "supervertex" while going down the search tree. Figure 1 shows the complete branching tree for bisecting a graph with $n = 6$ vertices into two components of size 4 and 2, i.e. $d = 2$. By "p, q" we denote that p and q are in the same set while "$p|q$" means that p and q are separated. A current (partial) solution can be represented by a vector consisting of "0","+" and "−", where a 0 entry means that the corresponding vertex has not been assigned yet, and elements with the same sign correspond to vertices in the same set.

Note that vertex 1 gets only a "+" in the tree and that we therefore seem to obtain solutions corresponding to $d < 0$ in Figure 1. But since we assume that the larger component is the first one, we consider the symmetric solution $(-x)$. The fixing of vertex 1 to "+" guarantees that only one of each pair of symmetric solutions is searched.

When we perform a switched merging we only change the sign of q since the new vertex becomes part of the supervertex. In Figure 1 we do not merge vertices explicitly, but give the sign pattern of the rows and columns of L which become a single row and column in the subproblem.

Summarizing, a subproblem at level k of the tree is obtained as follows. Depending on whether the subproblem results from its parent by a pure or a switched merging of vertices 1 and q, the cost matrix L and the weight vector w of the subproblem are obtained as described above. In the following, we will not use superscripts for L and w to indicate the level of the subproblem. This is done to simplify the notation.

3.2. **Bounding.** We now address various properties of the subproblems that have to be considered before we bound. These properties depend on the values of the quadruple (n, k, d, w_1), where k is the level of the subproblem in the search tree and w_1 is the weight of the supervertex. Knowing these values, we can investigate both structure and feasible solutions of a given subproblem.

Recall that the cardinalities of the two components into which the vertex set is partitioned are n_1 and n_2, respectively. The number of positive entries in the partial solution on which the subproblem is based is given by $p_+ := \frac{1}{2}((k+1)+w_1)$ while the number of negative entries equals $p_- := \frac{1}{2}((k+1)-w_1)$. We use $\mathcal{F}$ to denote the feasible set for the current subproblem

$$\mathcal{F} = \{x : x^t w = d,\ x \in \{-1,1\}^{n-k}\}. \tag{14}$$

We partition $\mathcal{F}$ into two sets $\mathcal{F} = \mathcal{F}_+ \cup \mathcal{F}_-$, with

$$\mathcal{F}_+ := \{x : x_1 = +1,\ x \in \mathcal{F}\} \text{ and } \mathcal{F}_- := \{x : x_1 = -1,\ x \in \mathcal{F}\}. \tag{15}$$

First of all we look at the case where we do not bound the subproblem but construct its feasible solutions explicitly and use the resulting objective function values as bounds. This is done if exactly one vertex can be arbitrarily assigned, while the other vertices are fixed. This situation occures if $\max\{p_+, p_-\} = n_1 - 1$ or $\min\{p_+, p_-\} = n_2 - 1$. In Figure 1, `(+++000)` is an example for the first and `(+-0000)` for the second case.

We bound a subproblem if $\max\{p_+, p_-\} < n_1 - 1$ and $\min\{p_+, p_-\} < n_2 - 1$ hold by solving relaxation (P^k). As argued above one has to prevent lack of interior points if $d = 0$. However, even if $d > 0$, there is a situation, where the resulting semidefinite relaxation (P^k) has no interior points, because the subproblem is equivalent to a certain equicut instance. This situation occurs when the first entry of each partial solution of the subproblem, $x \in \mathcal{F}$, becomes fixed, i.e. there are only solutions with x_1 being either positive or negative. This happens if $\max\{p_+, p_-\} > n_2$. If for instance $p_+ > n_2$, then $\mathcal{F}_-$ is empty, since $x^t w$ would become negative in the case $x_1 = -1$. The symmetric case deals with the situation

```
  L0              L1          L2       L3       L4

                                            /(+----0) = 1,6|2,3,4,5
                                    (+---00)
                                   /        \(+---+0) = 1,5|2,3,4,6
                           (+--000)
                           /       \        /(+--+-0) = 1,4|2,3,5,6
                          /         (+--+00)
                         /                  \(+--++0) = 1,4,5,6|2,3
               (+-00000)
              /          \                  /(+-+--0) = 1,3|2,4,5,6
             /            \         (+-+-00)
            /              \       /        \(+-+-+0) = 1,3,5,6|2,4
           /               (+-+000)
          /                        \        /(+-++-0) = 1,3,4,6|2,5
         /                          (+-++00)
        /                                   \(+-+++0) = 1,3,4,5|2,6
(000000)
        \                                   /(++---0) = 1,2|3,4,5,6
         \                          (++--00)
          \                        /        \(++--+0) = 1,2,5,6|3,4
           \               (++-000)
            \              /       \        /(++-+-0) = 1,2,4,6|3,5
             \            /         (++-+00)
              \          /                  \(++-++0) = 1,2,4,5|3,6
               (++00000)
                         \                  /(+++--0) = 1,2,3,6|4,5
                          \         (+++-00)
                           \       /        \(+++-+0) = 1,2,3,5|4,6
                           (+++000)
                                   \
                                    (++++00)          = 1,2,3,4|5,6
```

FIGURE 1. Branch-and-Bound Tree for Bisecting a Graph with $n = 6$ and $d = 2$.

that the number of $-$'s in the partial solution p_- is larger than n_2, so x_1 must be negative and hence $\mathcal{F}_+ = \emptyset$. But this means that if $p_+ > n_2$

$$\begin{aligned} \mathcal{F} = \mathcal{F}_+ &= \{x : x_1 = +1,\ x^t w = d,\ x \in \{-1,1\}^{n-k}\} \\ &= \{x : x_1 = +1,\ x^t \hat{w} = 0,\ \hat{w} = w - du_1,\ x \in \{-1,1\}^{n-k}\} \end{aligned}$$

and if $p_- > n_2$

$$\begin{aligned} \mathcal{F} = \mathcal{F}_- &= \{x : x_1 = -1,\ x^t w = d,\ x \in \{-1,1\}^{n-k}\} \\ &= \{x : x_1 = -1,\ x^t \hat{w} = 0,\ \hat{w} = w + du_1,\ x \in \{-1,1\}^{n-k}\}. \end{aligned}$$

Hence, the feasible set of the subproblem is equivalent to that of an equicut problem with a different weight vector and an extra constraint on x_1. By dropping this extra constraint in the equicut problem we obtain a relaxation of the subproblem which only adds symmetric solutions. Therefore the modified subproblem yields the same solution value as the original one.

If $d > 0$ and the subproblem is not equivalent to an equicut problem, it is given by

$$(16) \qquad (BIS^k) \qquad z^k := \min\{\tfrac{1}{4} x^t L x : x^t w = d,\ x \in \{-1,1\}^{n-k}\}.$$

In the equicut case or in the situation described above, the subproblem at the k-th level is given by

$$(17) \qquad (BIS^k) \quad z^k = \min\{\tfrac{1}{4}x^tLx : x^t\hat{w} = 0,\ x \in \{-1,1\}^{n-k}\}$$

where the weight vector is modified by $\hat{w} = w \pm du_1$, and if $d = 0$ we just solve the "original" equicut problem.

The semidefinite relaxations of (16) and (17) are obtained as described above. We denote the relaxation of (16) by (P_d^k) and the one of (17) by (P_0^k).

4. Algorithmical Aspects

In this section we describe how the lower bounds are calculated. We use a cutting plane approach based on semidefinite programming. For further details of this approach we refer to [17, 20, 25].

4.1. **The Primal-Dual Interior Point Approach.**

For solving problems (P_d^k) and (P_0^k) we employ the primal-dual interior point approach of Helmberg et al. [20]. Before we describe it, we show how one deals with the lack of strictly feasible points in the second relaxation (P_0^k). For ease of notation, let in the following n be the dimension of the current subproblem.

For relaxation (P_0^k) we need to project the problem in order to be able to apply the interior point approach, since every feasible solution has at least one eigenvalue equal to 0 with corresponding eigenvector $\hat{w}$. We introduce an $n \times (n-1)$ projection matrix V which contains a basis of the orthogonal complement of $\hat{w}$, i.e. for which $V^t\hat{w} = 0$ and $\operatorname{rank}(V) = n-1$ holds. Such a matrix can be easily found by setting

$$V = \begin{bmatrix} I_{n-1} \\ -(\hat{w}_{1:n-1})^t \end{bmatrix}.$$

Then we can substitute $X = VRV^t$ and obtain the following equivalent, projected problem to (P_0^k), whose matrix variable is of order $n-1$.

$$(\hat{P}_0^k) \quad \min\{\tfrac{1}{4}\mathrm{tr}\ V^tLVR : \operatorname{diag}(VRV^t) = e_n,\ \mathcal{B}(VRV^t) + b \geq 0,\ R \succeq 0\}.$$

Note that since V is full rank we have $X \succeq 0$ if and only if $R \succeq 0$.

In order to be able to describe the essential parts of the primal-dual interior point approach we introduce a unified model, which covers both (P_d^k) and $(\hat{P}_0^k)$. Suppose there are m_a equalities in a particular relaxation. We define a linear operator $\mathcal{A}(\cdot)$, which acts on the primal matrix variable and takes care of the equalities in the relaxation. The vector $a \in \Re^{m_a}$ corresponds to the constant part. For the general model (P_d^k) with $d > 0$ we have

$$\mathcal{A}(X) = \begin{pmatrix} \operatorname{diag}(X) \\ w^tXw \end{pmatrix}, \qquad a = \begin{pmatrix} e_n \\ d^2 \end{pmatrix}, \qquad m_a = n+1,$$

where the adjoint operator $\mathcal{A}^t(\cdot)$ acting on an m_a-vector is given by

$$\mathcal{A}^t(y) = \operatorname{Diag}(y_{1:n}) + y_{n+1}W, \qquad W := ww^t.$$

For the equicut case $(\hat{P}_0^k)$ the equalities are just

$$\mathcal{A}(X) = \operatorname{diag}(VXV^t), \qquad a = e_n, \qquad m_a = n,$$

where we now use X instead of R for ease of notation. The adjoint operator for the equality constraints acts on a vector $y \in \Re^{m_a}$ and is

$$\mathcal{A}^t(y) = V^t\operatorname{Diag}(y)V.$$

For the inequalities we use $\mathcal{B}(\cdot)$ as defined in Section 2.2, i.e. for the general and the equicut problem the r-th inequality is given by

$$\mathcal{B}_r(X) = \text{tr } H_{(r)}X, \text{ and } \mathcal{B}_r(X) = \text{tr } V^t H_{(r)} V X,$$

respectively. The constant part is just $b = -e$. The adjoint operators for the inequalities are

$$\mathcal{B}^t(u) = \sum_{r=1}^{m_b} H_{(r)} u_r, \text{ and } \mathcal{B}^t(u) = \sum_{r=1}^{m_b} V^t H_{(r)} V u_r$$

and act on a vector $u \in \Re^{m_b}$. Finally, we introduce a new cost matrix for the unified setting, which corresponds to $C = \frac{1}{4}L$ if $d > 0$, and to $C = \frac{1}{4}V^t L V$ if $d = 0$.

The primal-dual pair of the relaxations (in the unified setting) is now

$$(P) \quad \begin{array}{ll} \min & \text{tr } CX \\ \text{s.t.} & \mathcal{A}(X) + a = 0 \\ & \mathcal{B}(X) + b - s = 0 \\ & X \succeq 0, s \geq 0 \end{array} \qquad (D) \quad \begin{array}{ll} \max & a^t y - b^t u \\ \text{s.t.} & C + \mathcal{A}^t(y) - \mathcal{B}^t(u) - Z = 0 \\ & u - t = 0 \\ & Z \succeq 0, t \geq 0 \end{array}$$

where $s, t \in \Re^{m_b}$ are slack variables. Note that by weak duality, any feasible solution of (D) yields a lower bound for the graph bisection problem under consideration.

The Karush-Kuhn-Tucker optimality conditions for the dual log-barrier problem are

$$(KKT) \quad \begin{array}{rcccl} \mathcal{A}(X) + a & = & F_{P_1} & = & 0 \\ \mathcal{B}(X) + b - s & = & F_{P_2} & = & 0 \\ C + \mathcal{A}^t(y) - \mathcal{B}^t(u) - Z & = & F_{D_1} & = & 0 \\ u - t & = & F_{D_2} & = & 0 \\ ZX - \mu I & = & F_{ZX} & = & 0 \\ -t \circ s + \mu e & = & F_{ts} & = & 0 \end{array}$$

where the first and the second pair of conditions are for primal and dual feasibility, respectively. The last two equations are perturbed complementary slackness conditions. This system of (nonlinear) equations is solved using Newton's method. In each iteration the linearized form of (KKT) is solved for a fixed μ with respect to the correction $\Delta D = (\Delta y, \Delta u, \Delta X, \Delta Z, \Delta t, \Delta s)$. During the iterative process μ is driven to 0, yielding convergence to a solution of the primal-dual pair.

In the linearized (KKT)-conditions, $\Delta X, \Delta Z, \Delta s$ and Δt can be substituted and expressed in terms of Δy and Δu. This leads to a *final system*

$$(18) \quad \begin{array}{l} -\mathcal{A}(Z^{-1}\mathcal{A}^t(\Delta y)X) + \mathcal{A}(Z^{-1}\mathcal{B}^t(\Delta u)X) = -\mathcal{A}(Z^{-1}F_{D_1}X) - \mathcal{A}(Z^{-1}F_{ZX}) \\ -\mathcal{B}(Z^{-1}\mathcal{A}^t(\Delta y)X) + \mathcal{B}(Z^{-1}\mathcal{B}^t(\Delta u)X) - t^{inv} \circ (\Delta u) \circ s = \\ \qquad -\mathcal{B}(Z^{-1}F_{D_1}X) - \mathcal{B}(Z^{-1}F_{ZX}) + t^{inv} \circ F_{D_2} \circ s + F_{P_2} - t^{inv} F_{ts}. \end{array}$$

where $(t^{inv})_i = 1/t_i$. The final system (18) is of size $(m_a + m_b)$ and its solution is the most expensive part of the primal-dual algorithm. We have to construct an $(m_a + m_b) \times (m_a + m_b)$ matrix which represents the left hand side of (18), factorize it and solve for Δy and Δu. It can be shown that the resulting matrix is positive definite. The complexity for the construction and the solution using a Cholesky factorization is $\mathcal{O}(m_a + m_b)^3$. Finally, $\Delta X, \Delta Z$ and Δs are obtained by backsubstitution, and ΔX is symmetrized.

As in interior point approaches for linear programming, also in semidefinite programming predictor-corrector approaches prove to be very efficient, especially if the number of inequalities in the program becomes larger. We also employ this approach here and refer to the references above for details. We point out however that the final system (18) has to be factorized only once to obtain both the predictor and the corrector direction.

The last ingredient of the interior point method is the line search part to guarantee that the iterates stay in the interior of the feasible sets. In other words, we have to check whether the updated X and Z are positive definite which we do by performing Cholesky factorizations. The complexity of the line search is then $\mathcal{O}(n^3)$.

Global convergence of the primal-dual method was proved in [20], while the proof of quadratic convergence is due to [29]. In general, convergence to a fixed precision of say 10^{-5} is achieved in around $12-15$ iterations independent of the size of the semidefinite programming problem.

4.2. **The Cutting Plane Approach.** As already pointed out above, the number of possibly violated triangle inequalities is $\mathcal{O}(n^3)$. To obtain a relatively tight relaxation with a relatively small number of inequalities, we employ a cutting plane approach in the bounding procedure.

The cutting plane approach starts with solving relaxation (BIS_{SDP}) to the prespecified precision of convergence. We construct an upper bound from the solution and check whether we can fathom the node in the search tree. If not, we exhaustively search for violated triangle inequalities and add the n most violated inequalities as cutting planes. We call this a *large add* of inequalities.

Then we start the interior point method with a primal solution, which is in the strict interior of the currently feasible set and hence a strictly feasible point. After each iteration of the primal-dual method the duality gap becomes smaller, but at the same time we can not guarantee that the iterates are still in the interior of the feasible set with respect to the triangle inequalities. Therefore, we check after each iteration for violated inequalities. As soon as we find a violation, we could add the inequality as a cutting plane to the relaxation and restart the whole procedure from a strictly feasible point of the new relaxation. But in practice it is preferable to perform a few more iterations after a violation is detected, before adding new inequalities. Here, we make 3 more iterates, which results in a larger number of violated inequalities, of which we choose the $n/3$ most violated ones to add to the model. This is called a *small add*. Then we restart from an interior point of the current feasible set and check for newly violated inequalities after each iterate.

After 10 small adds we solve the current relaxation to optimality again and construct a feasible bisection and its corresponding upper bound, which are returned together with the lower bound to the main part of the branch-and-bound algorithm.

After this rather generic description of the bounding approach we specify a few algorithmical details. The degree of violation of an inequality is measured with respect to the barycenter of the feasible set, and not with respect to the origin. In other words, we measure the portion of the vector to the barycenter one has to move from the point violating a particular inequality until the inequality is satisfied with equality.

When there are no inequalities in the model or after performing a large add, we start from the barycenter $\hat{X}$ of the (primal) feasible set, whose construction is explained in the next section. After a small add, we just move back into the interior of the feasible set. This is done by taking a convex combination of the last point X_I which was strictly feasible with respect to all triangle inequalities and $\hat{X}$. Here we choose

$$X_S = .9X_I + .1\hat{X}$$

as new starting point. In the dual problem, we start feasible after a large add, and infeasible after a small add. We will address this issue in the next section, too.

We tested different combinations of large and small adds, but the setting with 1 large and 10 small adds turned out to be favorable. This setting also turned out to be effective for the max-cut problem [19].

Whenever the relaxation is solved to optimality we construct a feasible solution from X and improve it by local search. We first extract a column of X, say the q-th column X_q, which approximates $x \cdot x_q$, where x is a bisection, and set the n_1 largest elements to $+1$ and the others to -1. Then we use a simplified variant of the Kernighan-Lin heuristic [28] with limited depth search to improve the solution. This “rounding” and improvement is generally performed on 10 columns of X, and the best feasible solution is used as an upper bound. For a more detailed description of the Kernighan-Lin heuristic we also refer to Lengauer's book [30].

4.3. **Primal and Dual Starting Points.** This section describes how strictly feasible points can be obtained for the primal and the dual relaxation of any subproblem in the branching tree. This is necessary to apply an interior point method and their existence guarantees strong duality by Slater's constraint qualification. We also address the issue of starting from dual infeasible points after small adds. Suppose, we consider a subproblem at the k-th level specified by (n, k, d, w_1). For ease of notation, we nevertheless use n as dimension of the subproblems which actually corresponds to $n - k$.

4.3.1. *Primal Starting Points.* The key for getting a primal strictly feasible point is the barycenter $\hat{X}$ of the feasible set $\mathcal{T}$ of the subproblem, i.e.

$$\hat{X} := \tfrac{1}{N} \sum_{X \in \mathcal{T}} X \tag{19}$$

with $N := |\mathcal{T}| = |\mathcal{F}|$.

For computing $\hat{X}$ we partition $\mathcal{T}$ into two sets, namely $\mathcal{T} = \mathcal{T}_+ \cup \mathcal{T}_-$, with

$$\mathcal{T}_+ := \{X : X = xx^t,\ x \in \mathcal{F}_+\} \text{ and } \mathcal{T}_- := \{X : X = xx^t,\ x \in \mathcal{F}_-\}.$$

The barycenters of the two sets are denoted by $\hat{X}_+$ and $\hat{X}_-$, respectively, and we define

$$N_+ := |\mathcal{T}_+| = |\mathcal{F}_+| = \binom{n-1}{m_+} \text{ and } N_- := |\mathcal{T}_-| = |\mathcal{F}_-| = \binom{n-1}{m_-},$$

where m_+ and m_- are the number of possible (-1)'s in $x_{2:n}$ of $\mathcal{F}_+$ and $\mathcal{F}_-$, respectively. We observe that $m_+ = \frac{1}{2}(n - 1 + w_1 - d)$ and $m_- = \frac{1}{2}(n - 1 - w_1 - d)$.

In these terms, the barycenter can be written as

$$\hat{X} = \tfrac{1}{N}(N_+\hat{X}_+ + N_-\hat{X}_-). \tag{20}$$

Note that in the equicut case $d = 0$ we have $N_+ = N_-$ and the following considerations simplify accordingly.

When computing the entries of $\hat{X}_+$ and $\hat{X}_-$ we have to distinguish between the diagonal elements, the elements of the first row and column, and the remaining entries. The diagonal elements of the barycenters are clearly 1 since they are 1 in all feasible solutions.

For calculating element $(\hat{x}_+)_{1j}$ of the first row of $\hat{X}_+$ we observe that there are

$$\mu_+ = \binom{n-2}{m_+ - 1}$$

solutions $x \in \mathcal{F}_+$ for which $x_1 x_j = -1$, and

$$\pi_+ = N_+ - \mu_+$$

solutions for which $x_1 x_j = 1$. Hence

$$(\hat{x}_+)_{1j} = (\hat{x}_+)_{j1} = \tfrac{1}{N_+}(-\mu_+ + \pi_+) = \frac{d - w_1}{n-1}. \tag{21}$$

For the other entries $(\hat{x}_+)_{ij}$ with $i, j > 1, i \neq j$ we have

$$\mu_+ = 2\binom{n-3}{m_+ - 1}$$

solutions $x \in \mathcal{F}_+$ for which $x_i x_j = -1$, and

$$\pi_+ = N_+ - \mu_+$$

solutions for which $x_i x_j = 1$. This yields

$$(\hat{x}_+)_{ij} = \tfrac{1}{N_+}(-\mu_+ + \pi_+) = -\frac{n - 1 - (d - w_1)^2}{(n-1)(n-2)}. \tag{22}$$

It is easy to verify that $w^t \hat{X}_+ w = d^2$. Analogous considerations lead to

$$(\hat{x}_-)_{1j} = (\hat{x}_-)_{j1} = -\frac{d + w_1}{n-1} \tag{23}$$

and

$$(\hat{x}_-)_{ij} = -\frac{n - 1 - (d + w_1)^2}{(n-1)(n-2)}, \tag{24}$$

where one can also easily see that $w^t \hat{X}_- w = d^2$. The barycenter $\hat{X}$ is now obtained by using the above quantities in (20).

The following theorem shows that any strict convex combination of $\hat{X}_+$ and $\hat{X}_-$ is strictly feasible. In fact, both $\hat{X}_+$ and $\hat{X}_-$ have eigenvalues equal to 0, which can be observed using the ideas of the proof of the following theorem.

Theorem 4.1. *Let $\hat{X}_+$ and $\hat{X}_-$ be given as above. Then for all $\lambda \in (0,1)$, the point $\hat{X}_\lambda$ defined by*

$$\hat{X}_\lambda := \lambda \hat{X}_+ + (1-\lambda)\hat{X}_-$$

is strictly feasible for (P_d^k) if $d > 0$.

If $d = 0$, there is an $\hat{R}_\lambda$ which consists of the first $(n-1)$ rows and columns of $\hat{X}_\lambda$ such that $\hat{R}_\lambda$ is strictly feasible for $(\hat{P}_0^k)$.

The proof of this result is not complicated but rather technical. Therefore we omit it here and refer to the technical report [27].

The following corollary holds since $N_+ > 0$ and $N_- > 0$ follow from the conditions in Section 3.2 that we asserted for bounding a subproblem.

Corollary 4.2. *The barycenter $\hat{X}$ given by (20) is strictly feasible for (P_d^k), while $\hat{R}$, consisting of the first $(n-1)$ rows and columns of $\hat{X}$, is strictly feasible for $(\hat{P}_0^k)$.*

There remains one problem, the computation of N_+ and N_-. But this has only to be done for the problem in the root node, since the coefficients for the subproblems can be obtained from the coefficients of their parents in the search tree. Using superscripts to denote the level of the subproblem we have after a pure merging

$$N_+^{k+1} = (1 - \tfrac{m_+^k}{n-1})N_+^k \text{ and } N_-^{k+1} = \tfrac{m_-^k}{n-1}N_-^k$$

while after a switched merging we get

$$N_+^{k+1} = \tfrac{m_+^k}{n-1}N_+^k \text{ and } N_-^{k+1} = (1 - \tfrac{m_-^k}{n-1})N_-^k.$$

4.3.2. *Dual Starting Points.* Without inequalities in the model and after a large add we always start dual feasible. In the general case $d > 0$ we first set $u = t = e$ which makes t strictly feasible. Then we choose y such that

$$Z = C + \mathcal{A}^t(y) - \mathcal{B}^t(u) = C + \mathrm{Diag}(y_{1:n}) + y_{n+1}W - \mathcal{B}^t(u)$$

becomes positive definite. We use $y_{n+1} = 0$, and choose $y_{1:n}$ large enough so that Z becomes diagonally dominant. In the equicut case we proceed in the same way, except that $y \in \Re^n$, since we have

$$V^t\mathrm{Diag}(y_{1:n})V = \mathrm{Diag}(y_{1:n-1}) + y_nW_{1:n-1,1:n-1}.$$

After a small add it is favorable to start from a point which is infeasible with respect to

$$u - t = 0.$$

Let t_{old} and u_{old} be the portion of t and u before the small add, and let t_{new} and u_{new} correspond to the respective elements of the newly added inequalities, i.e.

$$t = \begin{pmatrix} t_{old} \\ t_{new} \end{pmatrix} \text{ and } u = \begin{pmatrix} u_{old} \\ u_{new} \end{pmatrix}.$$

The "old" parts are strictly feasible since they were before the small add. Nevertheless, we perturb those of their entries which are very small in order to prevent numerical difficulties. For the new inequalities we set $t_{new} = e$ and $u_{new} = 0$. This allows us to keep the latest y as new starting point and yields therefore the (up to a small change caused by the perturbation of u) same dual objective function value as the latest dual problem. In practice, t and u become dual feasible after a few steps of the interior point method, and then the lower bound provided by the dual objective function is a valid lower bound again.

5. Computational Results

5.1. **Implementational Details.** Before we are going to describe the numerical experiments we consider a few implementational details. The main routine of the branch-and-bound program was implemented in MATLAB. The cutting plane approach which provides the lower bounds was written in C and uses both BLAS and LAPACK routines, whenever possible. The compiling options are simply "–O" for full optimization and "+z" for producing position independent code, which is necessary for MEX interfaces. Our experiments were performed on the HP 9000/735 and the running times are reported in the format "hours:minutes:seconds".

In the branch-and-bound algorithm, a depth first search strategy is used to search the branching tree. As described in Section 3.1, branching is performed on a pair of vertices $(1, q)$. As q we choose the vertex, whose column is closest to a $(-1, 1)$-vector, i.e. whose column X_q minimizes the Euclidean norm of $|X_q| - e$ for all q. This branching rule turns out to work quite well in practice.

In the cutting plane approach we perform 1 large and 10 small adds, where we add n and $n/3$ violated inequalities, respectively. Hence, the number of inequalities m_b is bounded from above by $\frac{13}{3}n$. As long as there are not more than $2n$ cutting planes in the model, we do not perform the predictor-corrector approach, but compute the "simple" Newton search directions. This turns out to be more efficient.

Each relaxation is solved to an accuracy of 10^{-5}, i.e. convergence corresponds to a relative gap between primal and dual being smaller than this value. The tolerance for primal and dual feasibility is set to 10^{-6}. When the required duality gap is reached, but not the desired feasibility, usually one or two predictor-corrector steps without changing the barrier parameter μ are sufficient to achieve the prescribed feasibility. However, this is hardly ever necessary and there are at most two of these extra iterates necessary while bounding a problem.

5.2. **Numerical Experiments.** The goal of our experiments was to show that an exact approach based on semidefinite programming is a step towards the solution of problem instances of general graphs having more than 100 vertices. We investigated both exact and approximate solutions for an extensive number of graphs, which were made available to us from colleagues or were generated by ourselves. We divide the presentation into four subsections, the first providing an overview of the test problems under consideration. The other subsections present and discuss the numerical results for the different classes of problem instances.

5.2.1. *Overview.* We tested our approach on various classes of graphs which we either generated ourselves or obtained from the literature. The following subsections contain the results of our experiments. The set of randomly generated graphs in Subsection 5.2.2 was already used in [25, 26] and can be obtained from `http://www.imm.dk/~sk/eqp.d`. The second set of graphs considered in Subsection 5.2.3 stems from Brunetta, Conforti and Rinaldi [5] and is available at `ftp://ftp.math.unipd.it/pub/Misc/equicut`. Subsection 5.2.4 contains graphs due to [13, 24, 35] which come from real world applications.

The tables in the following sections read as follows: "graph" specifies the name of the instance, "n" is the number of vertices, and "dens" gives the density in percent. A column contains the optimal cut when labeled "opt" and a feasible solution when labeled "sol". In the latter case we provide a performance guarantee of the solution, given either as relative gap in percent or the absolute gap in the number of edges in column "gap". This means that the solution guarantees

$$\text{opt} \geq \text{sol} - \lfloor \max\{\tfrac{\text{sol}}{100}, 1\} \cdot \text{gap} \rfloor.$$

"$|B|$" gives the number of nodes bounded, and "time" shows the running time.

In Section 5.2.3 we also give the computation times obtained by Brunetta, Conforti and Rinaldi [5] with their approach on a SPARC 10/41. In these tables "SDP-time" denotes our results, while "LP-time" gives the running times reported in [5].

If not explicitly stated otherwise, we consider equicuts, i.e. the case $d = 0$. For the general case, d is some portion of the number of vertices of the graph.

graph	n	sol	gap	$\|B\|$	time	sol	gap	$\|B\|$	time
				$d = 0$				$d = n/2$	
ex36a	36	117	0%	1	3	85	0%	13	31
ex36b	36	118	0%	1	5	84	0%	1	5
ex36c	36	124	0%	1	2	90	0%	1	4
ex60a	60	367	0%	49	5:00	268	0%	79	9:56
ex60b	60	357	0%	37	4:50	259	0%	75	10:23
ex60c	60	343	0%	29	4:08	250	0%	49	7:15
ex84a	84	742	0%	287	2:02:55	548	0%	339	2:20:25
ex84b	84	771	0%	553	4:19:03	562	0%	77	28:41
ex84c	84	753	0%	619	4:18:15	556	0%	519	3:23:38
ex108a	108	1247	0%	5133	88:56:57	–	–	–	–
ex108b	108	1282	0%	8755	164:58:22	–	–	–	–
ex108c	108	1240	0%	1383	27:15:46	–	–	–	–
ex108a	108	1247	1%	49	1:56:31	915	1%	55	1:37:42
ex108b	108	1282	1%	77	3:00:18	937	1%	69	2:19:58
ex108c	108	1240	1%	13	34:49	918	1%	89	2:49:12
ex132a	132	1885	1%	55	4:45:51	1379	1%	33	2:34:35
ex132b	132	1883	1%	1	5:41	1403	1%	205	13:22:39
ex132c	132	1854	1%	5	27:16	1371	1%	81	5:35:40

TABLE 1. Randomly generated unweighted instances; dens $\in [47, 51]$%.

graph	n	sol	gap	$\|B\|$	time	sol	gap	$\|B\|$	time
				$d = n/6$				$d = n/12 + 1$	
ex36a	36	112	0%	3	10	114	0%	1	3
ex36b	36	111	0%	1	2	114	0%	1	3
ex36c	36	119	0%	1	4	121	0%	1	2
ex60a	60	351	0%	75	5:38	360	0%	49	5:18
ex60b	60	345	0%	87	10:47	352	0%	45	6:01
ex60c	60	332	0%	131	11:21	337	0%	23	3:25
ex84a	84	721	0%	1555	9:16:51	735	0%	801	5:03:53
ex84b	84	741	0%	183	1:19:51	760	0%	491	3:16:60
ex84c	84	727	0%	399	2:59:31	744	0%	751	4:57:35
ex108a	108	1207	1%	61	2:06:43	1235	1%	105	3:40:14
ex108b	108	1241	1%	179	6:21:04	1268	1%	129	4:35:31
ex108c	108	1205	1%	113	5:06:59	1228	1%	37	1:27:02
ex132a	132	1825	1%	109	8:59:19	1869	1%	295	21:12:41
ex132b	132	1829	1%	15	1:22:22	1867	1%	3	16:52
ex132c	132	1799	1%	69	5:04:09	1833	1%	1	5:56

TABLE 2. Randomly generated unweighted instances; dens $\in [47, 51]$%.

graph	n	sol	gap	$\lvert B\rvert$	time	sol	gap	$\lvert B\rvert$	time
			$d=0$				$d=n/2$		
ex36d	36	1426	0%	3	11	1030	0%	1	2
ex36e	36	1482	0%	1	4	1086	0%	23	40
ex36f	36	1454	0%	1	4	1065	0%	21	38
ex60d	60	4151	0%	5	45	3086	0%	101	10:48
ex60e	60	4154	0%	35	4:46	3082	0%	55	5:30
ex60f	60	4132	0%	77	8:29	3066	0%	85	8:43
ex84d	84	8152	0%	193	1:14:18	6081	0%	393	2:20:00
ex84e	84	8327	0%	513	3:43:58	6182	0%	231	1:14:04
ex84f	84	8264	0%	305	1:50:41	6171	0%	467	2:57:25
ex108d	108	13891	0%	8517	135:40:36	–	–	–	–
ex108e	108	13699	0%	2351	30:02:33	–	–	–	–
ex108f	108	13709	0%	461	6:56:49	–	–	–	–
ex108d	108	13891	1%	1	2:59	10357	1%	1	2:32
ex108e	108	13699	1%	1	26	10195	1%	1	2:55
ex108f	108	13709	1%	1	2:57	10296	1%	5	13:05
ex132d	132	20581	1%	1	5:30	15371	1%	1	5:31
ex132e	132	20618	1%	1	5:42	15338	1%	1	5:17
ex132f	132	20707	1%	1	5:38	15485	1%	1	5:20

TABLE 3. Randomly generated weighted instances; dens $\in \{99, 100\}\%$.

graph	n	sol	gap	$\lvert B\rvert$	time	sol	gap	$\lvert B\rvert$	time
			$d=n/6$				$d=n/12+1$		
ex36d	36	1378	0%	55	1:18	1400	0%	3	11
ex36e	36	1440	0%	45	1:04	1464	0%	43	56
ex36f	36	1398	0%	1	5	1426	0%	7	18
ex60d	60	4041	0%	263	20:32	4103	0%	69	5:32
ex60e	60	4041	0%	235	27:14	4115	0%	145	12:57
ex60f	60	4003	0%	115	13:12	4076	0%	29	04:33
ex84d	84	7906	0%	197	1:11:24	8065	0%	169	1:06:13
ex84e	84	8078	0%	1051	6:20:05	8244	0%	1072	6:47:43
ex84f	84	8034	0%	1501	7:52:55	8188	0%	871	4:43:50
ex108d	108	13494	1%	1	2:56	13767	1%	1	2:48
ex108e	108	13305	1%	1	2:51	13569	1%	1	2:54
ex108f	108	13306	1%	1	2:23	13571	1%	1	2:55
ex132d	132	20005	1%	1	5:30	20410	1%	1	5:42
ex132e	132	20024	1%	1	5:29	20447	1%	1	5:40
ex132f	132	20100	1%	1	5:23	20513	1%	1	5:46

TABLE 4. Randomly generated weighted instances; dens $\in \{99, 100\}\%$.

5.2.2. *Randomly Generated Instances.* These graphs were generated for testing purposes in [25, 26] and consist of two classes. The first class of graphs with labels "a-c" are unweighted pseudo-random graphs with uniform edge probability $p = 1/2$. The other group consists of weighted pseudo-random graphs with edge weights uniformly drawn from the interval $[0, 10]$. They are labeled by "d-f". The dimensions of all instances lie between $36 \leq n \leq 132$, and for each size and type we have three graphs. We do not give the densities in the tables, but they are in the interval $[47, 51]$ percent for the unweighted graphs, and either 99% or 100% for the weighted instances.

We solve all instances up to size $n = 84$ to optimality using four different settings for d. We also compute optimal equicuts for the instances with 108 vertices. The remaining problems are solved with a performance guarantee of 1%.

Tables 1 and 2 give the results of various partition sizes for the unweighted instances, and Tables 3 and 4 for the weighted graphs. Comparing the running times with those of the other graph classes in the next subsections shows empirically that the randomly generated instances of this type constitute the most difficult class. The solution times for the different dimension are: seconds for $n = 36$, minutes for $n = 60$, hours for $n = 84$, and days for $n = 108$.

In almost all cases, 1%-approximations for weighted graphs can be obtained in the root node. For unweighted instances, the solutions with this performance guarantee can be obtained in a few hours for most of the instances under consideration.

We observe that in general the solution times do not seem to depend on the value of d.

5.2.3. *Brunetta-Conforti-Rinaldi Library.* We also tested our approach on a library created by Brunetta, Conforti and Rinaldi [5], short *BCR-library*, which contains 250 instances with 20 to 80 vertices. The instances of the BCR-library fall into five categories.

The first class consists of *pure random instances.* In the generation, the density of the graphs was fixed first. Edges belonging to the graph received weights uniformly drawn from $[1, 10]$, the remaining edges got 0 weights.

The second group contains *planar grid instances*, which are named "hxkg". They represent a weighted $h \times k$ grid in the plane, where the edge weights have weights from 1 to 10, drawn from a uniform distribution. The resulting graphs have $n = hk$ vertices and $m = 2hk - h - k$ edges.

Toroidal grid instances belong to the third category of instances. They were generated in the same way as the planar grid problems, except that they have $n = hk$ vertices and $m = 2hk$ edges. These instances are denoted by "hxkt".

The next class of graphs are *mixed grid instances*, which are dense graphs. The edges of a planar grid got weights uniformly drawn from $[1, 100]$, and all the other edges weights uniformly drawn from $[1, 10]$. The names of these graphs are "hxkm".

The last group of graphs in the library are *instances with negative weights.* They were generated in the same way as the instances of the first class, except that half of the edges got weights from $[-10, -1]$, again drawn from a uniform distribution.

We solved all instances to optimality. Instead of presenting a long list of results we follow [5] and give a small representative sample of the results. We chose the same instances as Brunetta et al. and added a few larger ones to the lists. Our results are summarized in Tables 5 to 8.

graph	n	dens	opt	$\|B\|$	SDP-time HP 9000	LP-time SPARC 10
v0.90	20	10%	21	1	1	3
v0.00	20	100%	401	1	1	2
t0.90	30	10%	24	1	1	20
t0.50	30	50%	397	17	22	1:43
t0.00	30	100%	900	3	6	1:13
q0.90	40	10%	63	1	4	2:59
q0.80	40	20%	199	31	1:09	9:41
q0.30	40	70%	1056	23	1:02	1:05:42
q0.20	40	80%	1238	7	25	55:03
q0.10	40	90%	1425	13	41	54:06
q0.00	40	100%	1606	1	4	19:12
c0.90	50	10%	122	1	10	15:58
c0.80	50	20%	368	45	3:04	2:00:25
c0.70	50	30%	603	49	4:02	2:50:23
c0.30	50	70%	1658	51	2:44	5:45:42
c0.10	50	90%	2226	55	2:39	4:20:02
c0.00	50	100%	2520	43	2:20	2:51:01
c2.90	52	10%	123	1	12	34:02
c4.90	54	10%	160	15	1:39	49:15
c6.90	56	10%	177	3	30	52:23
c8.90	58	10%	226	71	8:46	n.a.
s0.90	60	10%	238	37	4:57	2:54:48

TABLE 5. Equicut of randomly generated instances from the BCR-library.

graph	n	dens	opt	$\|B\|$	SDP-time HP 9000	LP-time SPARC 10
10x2g	20	15%	6	1	1	6
5x6g	30	11%	19	1	3	1:01
2x16g	32	9%	8	1	4	1:48
18x2g	36	8%	6	1	2	4:22
2x19g	38	8%	6	49	55	13:08
5x8g	40	9%	18	1	2	14:11
3x14g	42	8%	10	5	18	20:56
5x10g	50	7%	22	1	9	47:23
6x10g	60	6%	28	57	5:19	4:59:54
7x10g	70	5%	23	61	9:17	n.a.

TABLE 6. Equicut of planar grid instances from the BCR-library.

The longest running times of about 32 and 16 minutes were obtained for the largest negative instance and the largest toroidal grid which are both of size $n = 80$. All the other running times are below 10 minutes of CPU time, and most of the instances are solved within 2 minutes. The easiest class of graphs with respect

graph	n	dens	opt	$\|B\|$	SDP time HP 9000	LP time SPARC 10
4x5t	20	21%	28	1	1	3
6x5t	30	14%	31	1	3	45
8x5t	40	10%	33	1	6	8:14
21x2t	42	10%	9	1	5	17:41
23x2t	46	9%	9	33	2:05	46:28
4x12t	48	9%	24	3	17	50:12
5x10t	50	8%	33	1	6	33:51
10x6t	60	7%	42	43	5:50	1:48:37
7x10t	70	6%	45	47	9:32	7:51:37
10x8t	80	5%	43	45	15:44	n.a.

TABLE 7. Equicut of toroidal grid instances from the BCR-library.

graph	n	dens	opt	$\|B\|$	SDP time HP 9000	LP time SPARC 10
2x10m	20	100%	118	1	1	3
6x5m	30	100%	270	1	1	28
2x17m	34	100%	316	21	29	3:55
10x4m	40	100%	436	1	2	5:15
5x10m	50	100%	670	1	2	53:32
4x13m	52	100%	721	5	34	1:35:02
13x4m	52	100%	721	5	34	1:25:05
9x6m	54	100%	792	1	12	n.a.
10x6m	60	100%	954	1	8	3:35:20
10x7m	70	100%	1288	1	14	35:21:37

TABLE 8. Equicut of mixed grid instances from the BCR-library.

graph	n	dens	opt	$\|B\|$	SDP time HP 9000	LP time SPARC 10
t0.n.10	30	90%	-301	3	7	1:27
t0.n.00	30	100%	-337	1	3	1:55
q0.n.70	40	30%	-298	17	50	52:53
q0.n.50	40	40%	-389	23	55	7:58
q0.n.40	40	30%	-450	1	6	20:21
q0.n.00	40	100%	-471	31	1:06	11:28
c0.n.00	50	100%	-829	67	4:17	26:54:07
s0.n.80	60	20%	-465	3	40	1:21:32
o0.n.80	80	20%	-690	115	31:48	n.a.

TABLE 9. Instances with negative weights from the BCR-library.

to our approach are the mixed grid instances whose maximal solution times were about 30 seconds.

A comparison of these results with the ones from Subsection 5.2.2 shows that unweighted graphs having a density of about 50% seem to be more difficult to partition with our approach than others. For weighted instances, dense graphs with uniformly generated edge weights are much harder to bisect than dense graphs based on grid structures.

To put our running times into perspective, we shortly discuss the computation times obtained by Brunetta et al. [5]. Their experiments were performed on a SPARC 10/41. In [12], the performance of various computers was measured using standard linear equation software. The numbers given in [12] indicate that on LINPACK benchmarks the HP 9000/735 is roughly 6 times faster than the SPARC 10/41. Brunetta et al. used a generic cutting plane approach based on linear programming relaxations which does not exploit sparsity of the underlying graphs. This is definitely a disadvantage of their method but our approach does not exploit sparsity either. We observe that their running times are in general in the order of hours when ours are in the order of minutes. It is difficult to compare the performance of different methods across different platforms, but the results indicate that the approach based on semidefinite programming is more efficient.

5.2.4. *Other Graphs from the Literature.* We also tested our approach on various classes of graphs from the literature. The first class of problems are graphs representing *de Bruijn networks* of dimensions $n \in \{32, 64, 128\}$. These networks are prominent interconnections networks for parallel computers. For references into this direction and results regarding the bisection of de Bruijn networks, we refer to Feldmann et al. [13]. The graph representing a de Bruijn network has $n = 2^k$ vertices and is $2k$-regular, where k is the *basis* of the network. The unweighted graphs are quite sparse and have therefore minimum cuts of low costs.

The next group of instances was introduced by Johnson, Mehrotra and Nemhauser [24] and are *compiler design instances.* Even though these instances were used for the more general min-clustering problem, see [14, 24], we use them here as graph bisection instances. The graphs are weighted.

Finally, we consider an other class of real world instances. They are *mesh instances* and arise from an application of the finite elements method in fluids, see [35]. Thereby the problem is the LU-factorization of the matrix of a linear system, which is a band matrix with two bands. It can be modeled as a graph bisection problem in an unweighted planar graph.

The results for the de Bruijn networks in Table 10 show that the running times are comparable to the ones in Subsection 5.2.2 for randomly generated graphs.

All but one of the compiler design instances can be solved in the root node within 20 seconds of CPU time, see Table 11. In the solution of instance cd47b, one has to go down the branching tree on a relatively long path containing subproblems with an absolute gap of 1 between lower and upper bound.

The mesh instances are the easiest instances considered here, see Table 12. For all but one graph we found the optimal solution already in the root node of the search tree. This is probably due to the fact that the graphs under consideration are planar and that for those graphs semidefinite programming relaxations provide tight approximations, see for instance [17] for further references on results for the max-cut problem. The running times are well below one minute for all instances with

graph	n	dens	sol	gap	$\|B\|$	time
debr5	32	12%	10	0	3	6
debr6	64	6%	18	0	55	7:49
debr7	128	3%	30	0	711	23:18:29
debr7	128	3%	30	1	195	8:53:20

TABLE 10. Equicut of DeBruijn networks.

graph	n	dens	opt	$\|B\|$	time
cd30a	30	13%	302	1	2
cd30b	30	13%	302	1	2
cd45	45	10%	760	1	7
cd47a	47	9%	426	1	10
cd47b	47	9%	580	35	1:52
cd61	61	10%	2176	1	20

TABLE 11. Equicut of compiler design instances.

graph	n	dens	opt	$\|B\|$	time
m4	32	10%	6	1	1
ma	54	5%	2	1	3
me	60	5%	3	1	4
m6	70	5%	7	1	37
mb	74	4%	4	1	28
mc	74	5%	6	1	46
md	80	4%	4	1	29
mf	90	4%	4	1	24
m1	100	3%	4	15	18:15
m8	148	2%	7	1	5:21

TABLE 12. Equicut of mesh instances.

$n \leq 90$ vertices, and the bisection of the larger graphs can be done within 20 minutes of CPU time. The mesh instances were also used for experiments by Brunetta et al. [5] and Ferreira et al. [14]. As mentioned above, the running times in [5] were obtained on a SPARC 10/41 using a generic linear programming based cutting plane approach. For all instances with $n \geq 60$ vertices the computation times were above one hour. In [14], a cutting plane approach for the more general node capacitated graph partitioning problem based on linear programming relaxations was presented which exploits sparsity of the graphs. The tests for this method were performed on a SUN 4/50, and the resulting CPU times are about three times larger than ours. The results in [12] indicate that the performance of the HP 9000/735 is about 10 times better than the performance of the SUN 4/50 on LINPACK benchmarks. A comparison of the different approaches shows that exploiting sparsity is crucial for linear programming approaches.

6. Conclusion

We presented a branch and bound approach based on semidefinite and polyhedral relaxations for the graph bisection problem and tested it extensively. The computational results indicate that the present approach solves bisection problems on general graphs with $80 - 90$ vertices efficiently. If the graphs are planar or if they base on grids, bisections can be obtained for larger graphs in very reasonable computation times. Our results also compare favorably to previously published ones, which were obtained with cutting plane methods based on linear programming relaxations.

Regarding the solution of substantially larger bisection problems, the present method has its limitations. As long as there are no alternatives to interior point methods for solving the semidefinite relaxations, exact bisections of general medium sized graphs with several hundred vertices are out of reach. Recall that each iteration of the interior point method requires the factorization of a dense system of equations, whose size equals the number of constraints in the relaxation. Neither sparsity nor structure in the data can be exploited in a satisfactory way in the current solution procedure.

Acknowledgments

We thank L. Brunetta, C. Helmberg, G. Rinaldi and R. Weismantel for making their test data available to us, and C. Helmberg for valuable comments on an earlier version of this paper.

References

[1] F. ALIZADEH. Interior point methods in semidefinite programming with applications to combinatorial optimization. *SIAM J. Optim.*, 5:13–51, 1994.

[2] C.C. ASHCRAFT and J.W.H. LIU. Using domain decomposition to find graph bisectors. Technical report CS-95-08, York University, North York, Canada, 1995.

[3] F. BARAHONA, M. GRÖTSCHEL, M. JÜNGER, and G. REINELT. An application of combinatorial optimization to statistical physics and circuit layout design. *Operations Research*, 36:493–513, 1988.

[4] R.B. BOPPANA. Eigenvalues and graph bisection. In *Proceedings of the 28th IEEE Annual Symposium on Foundations of Computer Science*, pages 280–285, 1987.

[5] L. BRUNETTA, M. CONFORTI, and G. RINALDI. A branch-and-cut algorithm for the equicut problem. *Math. Program.*, 78:243–263, 1997.

[6] T.N. BUI and B.R. MOON. Genetic algorithm and graph partitioning. *IEEE Trans. Comput.*, 45:841–855, 1995.

[7] J. CLAUSEN and J. LARSSON TRÄFF. Implementation of parallel branch-and-bound algorithms – experiences with the graph partition problem. *Ann. Oper. Res.*, 33:331–349, 1991.

[8] M. CONFORTI, M.R. RAO, and S. SASSANO. The equipartition polytope I: formulations, dimensions and basic facets. *Math. Program.*, 49:49–70, 1990.

[9] M. CONFORTI, M.R. RAO, and S. SASSANO. The equipartition polytope II: valid inequalities and facets. *Math. Program.*, 49:71–90, 1990.

[10] R. DIEKMANN, B. MONIEN, and R. PREIS. Using helpful sets to improve graph bisections. In D.F. Hsu, A.L. Rosenberg, D. Sotteau, editors, *Interconnection Networks and Mapping and Scheduling Parallel Computations*. DIMACS Series in Discrete Mathematics and Theoretical Computer Science Vol. 21, AMS, 57–73, 1995.

[11] W.E. DONATH and A.J. HOFFMAN. Lower bounds for the partitioning of graphs. *IBM Journal of Research and Development*, 17:420–425, 1973.

[12] J. DONGARRA. Performance of various computers using standard linear equations software. Technical report cs-89-85, Computer Science Department, University of Tennessee, Knoxvilee, USA, 1997.

[13] R. FELDMANN, B. MONIEN, P. MYSLIWIETZ, and S. TSCHÖKE. A better upper bound on the bisection width of the de Bruijn networks. In *STACS'97 Proceedings*, volume 1200, pages 511–522. LNCS, 1995.

[14] C.E. FERREIRA, A. MARTIN, C.C. de SOUZA, R. WEISMANTEL, and L.A. WOLSEY. The node capacitated graph partitioning problem: a computational study. *Math. Program.*, 81:229–256, 1998.

[15] A. FRIEZE and M. JERRUM. Improved approximation algorithms for max k-cut and max bisection. In *Proceedings of the 4th International IPCO Conference*, volume 920, pages 1–13. LNCS, 1995.

[16] M.X. GOEMANS and D.P. WILLIAMSON. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42:1115–1145, 1995. A preliminary version entitled ".878-Approximation Algorithms for MAXCUT and MAX2SAT" has appeared in the Proceedings of the 26th Annual ACM Symposium on Theory of Computing, 422–431, 1994.

[17] C. HELMBERG. *An interior point method for semidefinite programming and max-cut bounds.* PhD thesis, Graz University of Technology, Graz, Austria, 1995.

[18] C. HELMBERG, S. POLJAK, F. RENDL, and H. WOLKOWICZ. Combining semidefinite and polyhedral relaxations to integer programs. In *Proceedings of the 4th International IPCO Conference*, volume 920, pages 124–134. LNCS, 1995.

[19] C. HELMBERG and F. RENDL. Solving quadratic $(0, 1)$-problems by semidefinite programs and cutting planes. *Math. Program.*, 82(3):291–315, 1998.

[20] C. HELMBERG, F. RENDL, R. J. VANDERBEI, and H. WOLKOWICZ. An interior point method for semidefinite programming. *SIAM J. Optim.*, 6(2):342–263, 1996.

[21] C. HELMBERG, F. RENDL, and R. WEISMANTEL. A semidefinite programming approach to the quadratic knapsack problem. In *Proceedings of the 5th International IPCO Conference*, volume 1064. LNCS, 1996.

[22] C. HELMBERG and R. WEISMANTEL. Cutting plane algorithms for semidefinite relaxations. In P.M. Pardalos and H. Wolkowicz, editors, *Topics in Semidefinite and Interior-Point Methods.* Fields Institute Communications Series Vol. 18, AMS, 197–213, 1998.

[23] D.S. JOHNSON, C.R. ARAGON, L.A. McGEOCH, and C. SCHEVON. Optimization by simulated annealing: an experimental evaluation; part 1, graph partitioning. *Operations Research*, 37 (6):865–892, 1989.

[24] E.L. JOHNSON, A. MEHROTRA, and G.L. NEMHAUSER. Min-cut clustering. *Math. Program.*, 62:133–151, 1993.

[25] S.E. KARISCH. *Nonlinear approaches for the quadratic assignment and graph partition problems.* PhD thesis, Graz University of Technology, Graz, Austria, 1995.

[26] S.E. KARISCH and F. RENDL. Semidefinite programming and graph equipartition. In P.M. Pardalos and H. Wolkowicz, editors, *Topics in Semidefinite and Interior-Point Methods.* Fields Institute Communications Series Vol. 18, AMS, 77–95, 1998.

[27] S.E. KARISCH, F. RENDL, and J. CLAUSEN. Solving graph bisection problems with semidefinite programming. Technical Report DIKU-TR-97/9, Department of Computer Science, University of Copenhagen, 1997.

[28] B.W. KERNIGHAN and S. LIN. An efficient heuristic procedure for partitioning graphs. *Bell Syst. Tech. J.*, 49:291–307, 1970.

[29] M. KOJIMA, S. SHINDOH, and S. HARA. Interior-point methods for the monotone linear complementarity problem in symmetric matrices. *SIAM J. Optim.*, 7:86–125, 1997.

[30] T. LENGAUER. *Combinatorial Algorithms for integrated Circuit Layout.* John Wiley and Sons Ltd, Chichester, 1990.

[31] H. PIRKUL and E. ROLLAND. A Lagrangian based heuristic for uniform graph partitioning. Working paper, A. Gary Anderson Graduate School of Management, University of California, Riverside, USA, 1996.

[32] S. POLJAK and F. RENDL. Nonpolyhedral relaxations of graph-bisection problems. *SIAM J. Optim.*, 5:467–487, 1995.

[33] F. RENDL and H. WOLKOWICZ. A projection technique for partitioning the nodes of a graph. *Ann. Oper. Res.*, 58:155–180, 1995.

[34] E. ROLLAND, H. PIRKUL, and F. GLOVER. Tabu search for graph partitioning. *Ann. Oper. Res.*, 63, 1996.

[35] C.C. de SOUZA, R. KEUNINGS, L.A. WOLSEY, and O. ZONE. A new approach to minimising the frontwidth in finite element calculations. *Computer Methods in Applied Mechanics and Engineering*, 111:323–334, 1994.

[36] H. WOLKOWICZ and Q. ZHAO. Semidefinite programming relaxations for the graph partition problem. Research report CORR, Department of Combinatorics and Optimization, University of Waterloo, Waterloo, Canada, 1996.

Carmen Systems AB, Odinsgatan 9, S-411 03 Gothenburg, Sweden.
E-mail address: stefank@carmen.se

Department of Mathematics, University of Klagenfurt, Universitätsstrasse 65-67, A-9020 Klagenfurt, Austria.
E-mail address: franz.rendl@uni-klu.ac.at

Department of Mathematical Modelling, Bldg. 321, Technical University of Denmark, DK-2800 Lyngby, Denmark.
E-mail address: jc@imm.dtu.dk