Technical Report IMM-REP-1998-10
Department of Mathematical Modelling, Technical University of Denmark

CUTSDP — A TOOLBOX FOR A CUTTING–PLANE APPROACH BASED ON SEMIDEFINITE PROGRAMMING

USER'S GUIDE / VERSION 1.0

STEFAN E. KARISCH

ABSTRACT. CUTSDP is a package of C programs containing an implementation of a cutting plane approach based on semidefinite programming. This manual describes the package itself and also three applications: max-cut, graph bisection, and graph equipartition.

1. INTRODUCTION

This software package computes lower bounds for quadratic integer programming problems (QIP) in binary variables, i.e. in $\{-1,1\}$ or $\{0,1\}$ variables. The package combines a cutting plane approach with semidefinite programming relaxations of QIP where the semidefinite relaxations are of the form

$$(P)\quad \begin{array}{lll} \min & \operatorname{tr} CX & \\ \text{s.t.} & \operatorname{tr} A_i X = a_i & 1 \le i \le m_a \\ & X \succeq 0. & \end{array}$$

Let us denote by $\mathcal{S}_n$ the space of symmetric $n \times n$ matrices. Then $X \in \mathcal{S}_n$ is the matrix variable, $C \in \mathcal{S}_n$ the cost matrix, and $A_i \in \mathcal{S}_n$ are the constraint matrices in (P). There are m_a equality constraints with right hand side $a \in \Re^{m_a}$.

Note that for all relaxations (P) of QIP in $\{-1,1\}$ variables, the diagonal elements of X are one, i.e. the constraints

$$\operatorname{diag}(X) = (x_{11}, \dots, x_{nn})^t = e$$

are always present, where e is the vector of all ones. Moreover, there are also $\{0,1\}$ quadratic integer programming problems whose semidefinite relaxations contain diagonal constraints of this type.

In order to obtain good lower bounds, (P) can be tightened with polyhedral information by using generic and problem specific valid inequalities.

To solve (P), CUTSDP uses a primal-dual interior point approach based on the XZ-method [4, 9]. If (P) has no strictly feasible solutions, interior point approaches can not be readily applied. In CUTSDP, this issue is dealt with by eliminating the nullspace of the constraints and projecting (P) on a smaller dimensional space. This is done by introducing an $n \times \hat{n}$ matrix V, whose columns form a basis for

---

Date: June 12, 1998.

the $\hat{n}$-dimensional nullspace under consideration. Substituting $X = VRV^t$, the projected relaxation in the new variable $R \in \mathcal{S}_{\hat{n}}$ is given by

$$(\hat{P}) \quad \begin{array}{lll} \min & \mathrm{tr}\ CVRV^t & \\ \text{s.t.} & \mathrm{tr}\ A_i VRV^t = a_i & 1 \le i \le m_a \\ & R \succeq 0. & \end{array}$$

If we define $\hat{C} = V^t CV$, and $\hat{A}_i = V^t A_i V$, it is easy to see that $(\hat{P})$ is of the same form as (P).

In the Appendix, we describe the application of CUTSDP to three concrete examples: max-cut, graph bisection, and graph equipartition. For further examples of semidefinite relaxations to integer quadratic programming problems, we refer to eg. [1, 2, 3, 4, 5, 7, 8, 10].

1.1. **Why yet an other software package for SDP.** CUTSDP evolved out of several years of research on and implementation of semidefinite programming based bounds for two related types of combinatorial optimization problems, graph partition and quadratic assignment problems (see [6, 7, 8, 10]). CUTSDP contains the tools for applying semidefinite programming to other quadratic integer problems as well. From this point of view the package was not designed to serve as SDP solver per se. Other software packages are better suited for this purpose, see e.g. `http://www.zib.de/helmberg/semidef.html#Software`. But within the field of its application, CUTSDP constitutes to the author's knowledge the first publically accessible software package combining semidefinite programming and cutting plane approaches.

2. Obtaining and Installing CUTSDP

2.1. **CUTSDP home page.** CUTSDP is available via World Wide Web through the CUTSDP home page at

`http://www.imm.dtu.dk/~sk/cutsdp`

It contains this manual, a compressed tar file of the software package, and the latest news on CUTSDP.

To obtain CUTSDP, download the compressed tar file `cutsdp.tar.gz` with your browser.

2.2. **Installation.** To unpack the package type '`% gunzip cutsdp.tar.gz`' and then '`% tar xvf cutsdp.tar`', where "`%`" denotes the shell prompt. The package is composed of the following. The main directory `cutsdp` consists of the files `Makefile` and `README` and the subdirectories

- `source`: main routines of the toolbox `cutsdp.h`, `cut.c`, `cut_subs.c`, `sdp.c`, `sdp_subs.c`, `user_subs.c`.
- `manual`: this document `cutsdp.ps`.
- `applic`: subroutines for the max-cut problem `mc.c`, `mc_subs.c`; for the bisection problem `bis0.c`, `bis0_subs.c`, `bisd.c`, `bisd_subs.c`; and for the equipartition problem `eqp.c`, `eqp_subs.c`.
- `graphs`: graphs for testing purposes.
- `linalg`: BLAS and LAPACK routines used in CUTSDP in files `blas.f` and `lapack.f`.

Before compilation, the `Makefile` must be adapted to the architecture on which CUTSDP will be used. Check the file itself for a closer description and some examples of the various compiler options. The current setting always produces object files `lapack.o` and `blas.o` for the LAPACK or BLAS routines needed in CUTSDP. However, the user decides whether these files or the libraries are linked by defining `LINALG` in `Makefile`.

To compile the sample routines in CUTSDP, use the following commands.

'% make mc': builds the executable `mc` for the max-cut problem.

'% make bis0': builds the executable `bis0` for the graph bisection problem into equally sized components (i.e. the equicut problem).

'% make bisd': builds the executable `bisd` for the graph bisection problem into differently sized components.

'% make eqp': builds the executable `eqp` for the graph equipartition problem.

3. Semidefinite programming and cutting planes

3.1. **Basic relaxation and cutting planes.** In CUTSDP, (P) is considered to be the *basic relaxation* of QIP. Using a cutting plane approach, the relaxation is tightened to obtain better lower bounds for QIP. In the current version of CUTSDP, simple generic inequalities are used as cutting planes.

If QIP is given in $\{-1,1\}$ variables, CUTSDP provides the so called *triangle inequalities* as generic cutting planes. These inequalities are the simplest of the so called *hypermetric inequalities*, which base on the fact, that for any $x \in \{-1,1\}^n$, the inequality $h^t(xx^t)h \geq 1$ is valid, if $h \in \Re^n$ is integer and h^te is odd. Since the matrix variable X in relaxation (P) comes from defining $X := xx^t$, these inequalities are just

$$\text{tr } (hh^t)X \geq 1.$$

In case of the *triangle inequalities*, h contains only zeros except three elements which are either $+1$ or -1. They read,

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

taking symmetry and the diagonal constraints into account.

For QIP in $\{0,1\}$ variables, nonnegativity constraints are used as cutting planes, i.e. we add inequalities of the type

$$\text{tr } (u_iu_j^t)X \geq 0 \Longleftrightarrow x_{ij} \geq 0$$

where u_i is the i-th canonical unit vector.

But the user can also include additional valid inequalities by modifying subroutine `get_ineq()` in `source/cut_subs.c`. Details for doing this are described below.

3.2. **The cutting plane approach.** Cutting plane approaches combined with semidefinite relaxations were successfully applied for approximating various combinatorial optimization problems, see e.g. [3, 4, 5, 7, 8, 10]. We outline how this approach is used in CUTSDP.

The cutting plane approach starts with solving the basic relaxation (P) to the prespecified precision of convergence (`DUAL_GAP` in `source/cutsdp.h`). We might construct an upper bound from the solution and check whether the bound is tight. If this is not the case, we exhaustively search for violated (generic and/or user defined) inequalities and add the (`LARGE_INEQ`$*n$) 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. (Note that the method implemented in CUTSDP works best if the starting points are well centered.) After each iteration of the primal-dual method the duality gap is reduced, but at the same time it is very likely, that the iterates violate some of the polyhedral inequalities. Therefore, we check after each iteration for violated inequalities. As soon as a violation is detected, 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. CUTSDP makes `MAX_EXTR` more iterates, which results in a larger number of violated inequalities, of which we choose the (`SMALL_INQ`$*n$) 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 a prespecified number of small adds the current relaxation is solved to optimality again, and one might construct a feasible solution and its corresponding upper bound. This process can be repeated by making a new large add followed by a number of small adds. Before performing a new large add, inequalities whose dual costs w_i are relatively small, are removed. To be more specific, inequality i is discarded, if $w_i <$ `REM_INEQ` $* \, w_{\max}$ with $w_{\max} = \max_j\{w_j\}$, holds.

3.3. **The primal-dual interior point approach.** For solving the semidefinite relaxations of QIP we employ the primal-dual interior point approach of [4, 9]. In this subsection we shortly outline the approach and refer to parameters which can be set in CUTSDP. Global convergence of the primal-dual method was proved in [4], while the proof of quadratic convergence is due to [9].

At the beginning and after each add of inequalities, the following primal-dual pair of semidefinite programs has to be solved

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

with m_a(`ma`) equalities and m_b(`mb`) inequalities. The equality constraints are represented by $\mathcal{A}(\cdot) : \mathcal{S}_n \to \Re^{m_a}$ and $a \in \Re^{m_a}$, while the inequality constraints corresponding to the cutting planes are given by $\mathcal{B}(\cdot) : \mathcal{S}_n \to \Re^{m_b}$, $b \in \Re^{m_b}$. $s, t \in \Re^{m_b}$ are the primal and dual slack variables, respectively. Note that by weak duality, any feasible solution of $(DSDP)$ yields a lower bound for the QIP under consideration.

The Karush-Kuhn-Tucker optimality conditions for the primal-dual pair are

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

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 $\Delta D = (\Delta y, \Delta w, \Delta X, \Delta Z, \Delta t, \Delta s)$. During the iterative process μ is driven to 0, yielding convergence to a solution of the primal-dual pair. This iterative process is terminated when one of the following stopping criteria is met: either the necessary duality gap `DUAL_GAP` and amount of feasibility `FEAS_TOL` are reached, the step sizes become too small (i.e. smaller than `STEP_TOL`), or the number of iterations becomes too large (i.e. larger than `MAX_ITER`).

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

$$(2)\qquad \begin{array}{l} -\mathcal{A}(Z^{-1}\mathcal{A}^t(\Delta y)X)+\mathcal{A}(Z^{-1}\mathcal{B}^t(\Delta w)X) = -\mathcal{A}(Z^{-1}F_{D_1}X)-\mathcal{A}(Z^{-1}F_{ZX})\\ -\mathcal{B}(Z^{-1}\mathcal{A}^t(\Delta y)X)+\mathcal{B}(Z^{-1}\mathcal{B}^t(\Delta w)X)-t^{inv}\circ(\Delta w)\circ s =\\ \qquad -\mathcal{B}(Z^{-1}F_{D_1}X)-\mathcal{B}(Z^{-1}F_{ZX})+t^{inv}\circ F_{D_2}\circ s+F_{P_2}-t^{inv}F_{ts}. \end{array}$$

where $(t^{inv})_i = 1/t_i$. The final system (2) is of size (m_a+m_b) and its solution is the most expensive part of the primal-dual algorithm. An $(m_a+m_b)\times(m_a+m_b)$ matrix representing the left hand side of (2) has to be constructed, factorized and solved for Δy and Δw. 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 back substitution, 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. This approach is also employed in CUTSDP. For details, the reader is referred to the references given above. We point out however that the final system (2) has to be factorized only once to obtain both the predictor and the corrector direction. Within CUTSDP the parameter `PRED_COR` determines when predictor-corrector search directions are computed. If $m_b \geq$ `PRED_COR` $* n$, then a predictor-corrector approach is used.

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 $\mathcal{O}(n^3)$. As in other implementations, we never make a full step to the boundary, but only `STEP_POR` of the steplength, where the default of `STEP_POR` is 0.9.

The reduction of μ is done in the following way. If the feasibility of the primal or the dual iterate is not within `FEAS_TOL` or the primal and dual stepsize is smaller than 0.2, then μ is not changed. Otherwise, it is halfed. If in addition, both primal and dual stepsize are larger than 0.9, then μ is halfed again, i.e. its new value is a quarter of its old.

3.4. **Summarizing the parameters.** The following summarizes the parameters defined in `source/cutsdp.h` and their default setting in CUTSDP.

```
/* problem specific                                                     */

#define ZEROONE       0  /* ...=1 if {0,1}, ...=0 if {-1,1}             */
#define DAT_INT       1  /* ...=1 if costs are integral                 */
#define PROJ_E        1  /* ...=1 if V'e=0                              */
#define PPAR          2  /* {0,1,2} print status                        */
#define UBD        1e10  /* (default) initial upper bound               */
#define FEASOL        0  /* no feasible solution method implemented     */

/* cut()                                                                */

#define MAX_INEQ   2000  /* max no of overall inequalities              */
#define LARGE_INQ     1  /* max ineqs at large add = LARGE_INQ*no       */
#define SMALL_INQ     3  /* max ineqs at small add = SMALL_INQ*no       */
#define MAX_EXTR      3  /* max no of exterior iterations before small add */
#define REM_INEQ   1e-4  /* remove ineq. if (dual_cost>REM_INEQ*max_cost)  */
#define OUTPUT        1  /* writes "output.m" if value equals 1         */

/* sdp()                                                                */

#define MAX_ITER    100  /* max no of iterations for solving sdp        */
#define DUAL_GAP  1e-05  /* duality gap needed for convergence          */
#define PRED_COR      1  /* computes pred/corr steps if mb >= PRED_COR*n   */
#define STEP_POR    0.9  /* portion of step towards the boundary        */

#define REL_TOL   1e-10  /* relative tolerance                          */
#define ABS_TOL   1e-10  /* absolute tolerance                          */

#define STEP_TOL  1e-06  /* tolerance for steplengths                   */
#define FEAS_TOL  1e-06  /* tolerance for feasibility                   */
#define FEAS_BND  1e+06  /* bound for infeasibility                     */
#define VIOL_TOL  1e-10  /* violation tolerance for inequalities        */
```

`ZEROONE` identifies, whether QIP is given in $\{-1,1\}$ or $\{0,1\}$ variables. If `DAT_INT` equals 1, lower bounds are rounded to the nearest integer towards infinity. Parameter `PROJ_E` determines, whether the projection matrix V eliminates row sum constraints on the matrix variable X, i.e. whether the columns of V form a basis for the orthogonal complement of the vector of all ones e. `PPAR` determines the output of CUTSDP: 0 is silent mode, 1 is default, and 2 provides full information. `UBD` contains an initial upper bound, and `FEASOL` is set to 1 if `get_feas()` has been implemented.

`MAX_INEQ` is the upper bound for m_b. The numbers of inequalities added at a large and a small add are determined by `LARGE_INQ` and `SMALL_INQ`, respectively. `MAX_EXTR` gives the number of extra iterations the interior point method performs before a small add is done. `OUPTUT` indicates whether an output file `output.m` is written.

The remaining parameters have been already explained above or are self-explaining tolerance parameters.

Note that not all parameters are read from `source/cutsdp.h`, but that some are passed when calling the main routine `cut()`.

4. Input data and storage conventions

The input data for a certain problem type are the cost matrix C and the constraints defined by A_i and a_i. Moreover, the user might need to provide information about the projection matrix V, too. Note, that since the main routines of CUTSDP are implemented in C, in the following the indices start with 0, e.g. the index set of a vector of length n is $\{0, \ldots, n-1\}$.

All matrices are stored in column fashion, i.e. their columns are packed to one vector. Hence the cost matrix C is stored as one-dimensional array `C` of length n^2. Both equalities and inequalities are stored in the same fashion. We describe the storage conventions for the equality constraints. First, `ma` gives the number of constraints and `a` is an array of length m_a representing the right hand side a.

The arrays `ranka`, `densa` and `numba` of length m_a indicate how the constraint matrices A_i are stored. Entry i of the first array indicates the rank of A_i: the current values implemented are 1 for rank one constraints $A_i = a_i a_i^t$, and 3 for general constraints. A future version of CUTSDP will also consider rank-two constraints. The entries in `densa` indicate whether the constraints are to be considered dense or sparse, e.g. `densa[i] = 1` if constraint i is stored in a dense fashion, the value is 0 for a sparse constraint. Finally, `numba` gives the number of elements used to describe the constraints.

`ira` and `ica` are one-dimensional arrays containing the indices of the stored entries of the constraint (which correspond to the locations of the nonzeros in sparse storage). `A` is a field containing all the information of the A_i's in stacked form, i.e., all the entries specified by `ira` and `ica`.

Let us consider the following simple example of storing the $m_a = n$ constraints

$$\operatorname{diag}(X) = \begin{pmatrix} \operatorname{tr} E_{00}X \\ \vdots \\ \operatorname{tr} E_{n-1,n-1,}X \end{pmatrix} = \begin{pmatrix} \operatorname{tr} u_0 u_0^t X \\ \vdots \\ \operatorname{tr} u_{n-1} u_{n-1}^t X \end{pmatrix} = e.$$

Let us look at the i-th constraint in more detail. We have: `a[i] = 1`. If the constraint is stored as rank one constraint, then `ranka = 1`. It also make sense to store it sparse, since it has only one nonzero entry. Hence `densa[i] = 0` and `numba[i] = 1`. If all the previous $i-1$ constraints are stored in the same manner, then `ira[i] = i`, `ica[i]` is not needed, and `A[i] = 1`.

The other extreme for storing these constraints are a full rank and dense fashion. This yields: `ranka[i] = 3`, `densa[i] = 1` and `numba[i] = n*n`; and for entry (i', j') of A_i

$$\mathtt{ira}[\mathtt{i}*\mathtt{n}*\mathtt{n}+\mathtt{i}'*\mathtt{n}+\mathtt{j}'] = \mathtt{i}'$$

$$\mathtt{ica}[\mathtt{i}*\mathtt{n}*\mathtt{n}+\mathtt{i}'*\mathtt{n}+\mathtt{j}'] = \mathtt{j}'$$

$$\mathtt{A}[\mathtt{i}*\mathtt{n}*\mathtt{n}+\mathtt{i}'*\mathtt{n}+\mathtt{j}'] = \begin{cases} \mathtt{1} & \text{if } \mathtt{i}' = \mathtt{j}' \\ \mathtt{0} & \text{otherwise} \end{cases}$$

If `PROJ_E = 0` and a projection matrix V is needed, then V is stored in a one dimensional field of length $n * \hat{n}$. If `PROJ_E = 1` then V is not needed.

5. General routines

The main routines of CUTSDP are `cut()`, `cut_subs()`, `sdp()`, and `sdp_subs()`. The first one is the main routine of the package through which CUTSDP is called.

5.1. Calling `cut()`. The following shows how `cut()` is invoked.

```
long cut(
   double *Xfeas,    /* integer primal feasible solution */
   long no,          /* order of C */
   double *C,        /* cost matrix */
   long ma,          /* number of equalities */
   double *a,        /* equality constraints */
   long mb,          /* number of inequalities */
   double *b,        /* inequality constraints */
   long *ranka,      /* rank of equality constraints */
   long *densa,      /* density of equality constraints */
   long *numba,      /* number of equality constraints */
   long *ira,        /* row indices */
   long *ica,        /* column indices */
   double *A,        /* equality coefficients */
   long *rankb,      /* rank of inequality constraints */
   long *densb,      /* density of inequality constraints */
   long *numbb,      /* number of inequality constraints */
   long *irb,        /* row indices */
   long *icb,        /* column indices */
   double *B,        /* inequality coefficients */
   long lengthB,     /* possible number of entries of B */
   double *V,        /* projection information */
   long proj_e,      /* ... = 1 if V'e = 0 */
   long n,           /* order of projected matrices */
   double *prblminf, /* problem specific information */
   long zeroone,     /* {0,1} or {-1,1} */
   double *ubd,      /* upper bound */
   double *lbd,      /* lower bound */
   long maxlarge,    /* maximal number of large adds */
   long maxsmall,    /* maximal number of small adds */
   long ppar,        /* print various info yes/no */
   long dat_int,     /* costs integral (==1) or not (==0) */
   long feasol       /* feasible solution method implemented? */
   )
```

Most of the parameters are self-explaining or have already been described above, therefore we discuss only a few here.

First, let us look at parameters which correspond to some which are defined in `source/cutsdp.h`. These are `proj_e`, `ubd`, `zeroone`, `ppar`, `dat_int`, and `feasol` and they correspond to the parameters with same names but in capital letters. This set of parameters in `source/cutsdp.h` is not accessed within CUTSDP itself, because they contain problem specific information. Since the user should not have to recompile the whole package to be able to change their values, we decided to pass them when `cut()` is invoked. However, if they should take their default values given in `source/cutsdp.h`, the user can easily access these values from the calling routine. For more details about these parameters, we refer the reader to implementational issues of the applications in the appendix.

Most of the variables are input parameters. `no` corresponds to the dimension of the QIP, while `n` is the dimension of the projected relaxation, i.e. the number of columns $\hat{n}$ of the projection matrix V. `lengthB` is the reserved length of the one-dimensional field `B` containing the coefficients for the inequality constraints. `prblminf` contains problem specific information for the QIP under consideration which is not covered by any of the other parameters. In a graph bisection problem for example, it would contain information on the cardinality of the components into which the vertices of the graph are to be partitioned (for more information see the Appendix below). Note, that `cut()` does not check the input for its correctness!

Output parameters are `Xfeas` and `lbd`. `Xfeas` is an array of length n^2 which contains an integer primal feasible solution in the case that `FEASOL=1`. `lbd` gives the lower bound found.

The parameter `info`, which is returned by `cut()` contains the termination status of CUTSDP. Its possible values are as follows:

```
info =  0 ... successful termination
info =  1 ... primal indefinite (singular)
info =  2 ... dual indefinite (singular)
info =  3 ... primal slack nonpositive
info =  4 ... dual slack nonpositive
info =  5 ... final system indefinite
info =  6 ... steplengths become to small
info =  7 ... maxit reached
info =  8 ... memory allocation failed
info = -1 ... primal infeasible
info = -2 ... dual infeasible
```

Furthermore, if `OUTPUT` is set to 1, then `cut()` writes a file `output.m` which contains, depending on the parameter setting, the variables `ubd`, `lbd`, `Xfeas`, `X`, and `C` in Matlab format.

6. User provided routines

There are routines that must be provided by the user, and others that can be modified for specific applications.

The following routines are invoked from either `cut()` or `sdp()` and their headers are contained in the file `source/user_subs.c`.

```
void barycent( /* barycenter of the primal feasible set            */
   long n,          /* order of matrix */
   double *Xhat,    /* barycenter */
   double *prblminf /* problem specific information */
   )
```

`barycent()` is the routine which provides CUTSDP with a strictly feasible primal solution. The package works best if `Xhat` is the barycenter of the feasible set of QIP. `Xhat` is of length $\hat{n}^2$.

```
void dualfeas( /* computes strictly dual feasible points          */
   long n,          /* order of matrix */
   double *Z,       /* dual matrix */
   long no,         /* order of unprojected matrices */
   double *C,       /* cost matrix */
   long ma,         /* number of equalities */
   double *y,       /* dual variable */
```

```
   long mb,          /* number of inequalities */
   double *w,        /* dual variable */
   double *t,        /* dual slack variable */
   long *ranka,      /* rank of equality constraints */
   long *densa,      /* density of equality constraints */
   long *numba,      /* number of equality constraints */
   long *ira,        /* row indices */
   long *ica,        /* column indices */
   double *A,        /* equality coefficients */
   long *rankb,      /* rank of inequality constraints */
   long *densb,      /* density of inequality constraints */
   long *numbb,      /* number of inequality constraints */
   long *irb,        /* row indices */
   long *icb,        /* column indices */
   double *B,        /* inequality coefficients */
   double *V,        /* projection information */
   long proj_e,      /* ... = 1 if V'e = 0 */
   double *prblminf, /* problem specific information */
   double *H,        /* workspace */
   double *H1,       /* workspace */
   double *h,        /* workspace */
   double *h1        /* workspace */
   )
```

This subroutine provides `sdp()` with a strictly dual feasible solution consisting of (y, w, t, Z). The workspace variables `H` and `H1` are arrays of length n^2, while `h` and `h1` are of length n.

```
void get_feas( /* computes an integer feasible solution out of X  */
   long n,           /* order of matrix */
   double *X,        /* primal matrix */
   long no,          /* order of unprojected matrices */
   double *C,        /* cost matrix */
   double *Xfeas,    /* integral primal solution */
   double *cubd,     /* upper bound */
   double *V,        /* projection information */
   long proj_e,      /* ... = 1 if V'e = 0 */
   double *prblminf, /* problem specific information */
   double *H,        /* workspace */
   double *H1        /* workspace */
   )
```

Subroutine `get_feas` is only called from `cut()` if `feasol=1`. In this case `cubd` is an upper bound which must be provided by the subroutine. The other parameters are identical to the ones discussed above.

If not only the generic inequality constraints should be used by `CUTSDP` the user can achieve this by changing subroutine `get_ineq()` in `source/cut_subs.c` appropriately. In the source code we marked the locations where the changes should be done by '`*** BEGIN CHANGES ***`' and '`*** END CHANGES ***`'.

Acknowledgments

Most of this work was done while the author was at the Deparment of Mathematical Modelling at the Technical University of Denmark. The author thanks

Jesper Larsen and Franz Rendl for valuable comments on an earlier version of this package.

References

[1] F. ALIZADEH. Interior point methods in semidefinite programming with applications to combinatorial optimization. *SIAM J. Optim.*, 5:13–51, 1995.

[2] 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.

[3] C. HELMBERG and F. RENDL. Solving quadratic (0, 1)-problems by semidefinite programming and cutting planes. Preprint SC-95-35, Konrad-Zuse-Zentrum Berlin, Berlin, Germany, 1995.

[4] 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.

[5] C. HELMBERG, F. RENDL, and R. WEISMANTEL. A semidefinite programming approach to the quadratic knapsack problem. In *Integer Programming and Combinatorial Optimization*, volume 1064, pages 175–189. LNCS, 1996.

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

[7] 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, AMS, Providence, Rhode Island, 1995. To appear.

[8] 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, Copenhagen, Denmark, 1997.

[9] 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.

[10] Q. ZHAO, S.E. KARISCH, F. RENDL, and H. WOLKOWICZ. Semidefinite programming relaxations for the quadratic assignment problem. *Journal of Combinatorial Optimization*, 1996. To appear.

Appendix: Applications

In the following we describe three applications for CUTSDP which are provided with the software package. These are *max-cut*, *graph bisection*, and *graph equipartion*. We first introduce them shortly and give their basic relaxations. Then we describe how the calling routine was implemented for the package.

Max-cut. The max-cut problem can be formulated in the following way. Given an undirected edge weighted graph with n vertices. Find a partition of the vertex set into two components, such that the total weight of the edges cut by the partition, i.e. edges having their endpoints in different components, becomes maximal. Assuming that the weights are given trough the *Laplace matrix* L and using a vector $x \in \{-1,1\}^n$ as variable the problem can be formulated as the QIP:

$$(MC) \quad \begin{array}{ll} \max & \frac{1}{4}x^t L x \\ \text{s.t.} & x \in \{-1,1\}^n. \end{array}$$

Its basic relaxation in the form (P) can be obtained by setting $X = xx^t$ and $C = -\frac{1}{4}L$:

$$\begin{array}{ll} \min & \operatorname{tr} CX \\ \text{s.t.} & \operatorname{diag}(X) = e \\ & X \succeq 0. \end{array}$$

For further details into this direction we refer to e.g. [3].

Graph bisection. In the graph bisection problem, the task is to partition the graph into components of prescribed cardinality such that the weight of the edges cut becomes minimal. Given a parameter d which contains the difference of the number of vertices in the two sets, the problem can be written as a QIP in the following form:

$$(BIS)\quad \begin{array}{ll} \min & \frac{1}{4}x^tLx \\ \text{s.t.} & x^te = d \\ & x \in \{-1,1\}^n. \end{array}$$

The semidefinite relaxation in the form (P) is:

$$\begin{array}{ll} \min & \mathrm{tr}\ CX \\ \text{s.t.} & \mathrm{diag}(X) = e \\ & e^tXe = d^2 \\ & X \succeq 0. \end{array}$$

The difference between the cardinalities of the components d is passed trough `prblminf`.

If $d = 0$ there are no strictly feasible solutions for this relaxation. Therefore, the constraint $e^tXe = 0$ is eliminated by projection. We introduce $V \in \Re^{n\times(n-1)}$ such that $V^te = 0$, and the relaxation becomes

$$\begin{array}{ll} \min & \mathrm{tr}\ V^tCVR \\ \text{s.t.} & \mathrm{diag}(VRV^t) = e \\ & R \succeq 0. \end{array}$$

In this case `no`$= n$ and `n`$= \hat{n} = n - 1$.

For further details into this direction we refer to [8].

Graph equipartition. In the graph equipartion problem the graph under consideration must be partitioned into k equally sized components and the weight of the edges connecting different components is again minimized. Therefore we assume that n is a multiple of k. The equipartion problem can be modeled as $\{0,1\}$ QIP using an $n \times k$ matrix Y:

$$(EQP)\quad \begin{array}{ll} \min & \frac{1}{2}Y^tLY \\ \text{s.t.} & Y^te_n = \frac{n}{k}e_k \\ & Ye_k = e_n \\ & Y \in \{0,1\}^{n\times k}. \end{array}$$

The basic relaxation, where $C = \frac{1}{2}L$, $VRV^t = YY^t$, and equality constraints have been eliminated using the same projection as above, is:

$$\begin{array}{ll} \min & \mathrm{tr}\ V^tCVR \\ \text{s.t.} & \mathrm{diag}(VRV^t) = \frac{k-1}{k}e \\ & R \succeq 0. \end{array}$$

The number of the components K is passed through `prblminf`, and here we have again `no`$= n$ and `n`$= \hat{n} = n - 1$.

For further details into this direction we refer to [7].

Implementation in CUTSDP. In this subsection we specify how the applications described above are implemented in detail. The main programs are `applic/mc.c` for the max-cut problem, `applic/bisd.c` for the bisection problem with $d > 0$, `applic/bis0.c` for the equicut problem ($d = 0$), and `applic/eqp.c` for the equipartition problem. The files `applic/xxx_subs.c` contain the respective user provided subroutines needed for CUTSDP. The directory `graphs` contains sample instances of graphs that were generated for testing purposes in [6, 7, 8]. The graphs are given as adjacency lists, i.e. the first row contains the number of vertices and the number of edges, and the other rows constitue an edge list (i, j, w_{ij}), where w_{ij} is the weight of edge (i, j).

After compilation of the applications, see Section 2.2, the calling sequences are:

‘`% mc <graph> [maxlarge [maxsmall]]`’,
‘`% bisd <graph> <d> [maxlarge [maxsmall]]`’,
‘`% bis0 <graph> [maxlarge [maxsmall]]`’,
‘`% eqp <graph> <k> [maxlarge [maxsmall]]`’.

If the number of large adds of inequalities `maxlarge`, or the number of small adds of inequalities `maxsmall` are not specified they are set to 0. For example, the call ‘`% bis0 graphs/ex60a`’ executes the computation of the basic relaxation of the equicut problem for the instance `ex60a`, while ‘`% bis0 graphs/ex60a 3 2`’ runs CUTSDP with 3 large and 2 small adds. In the second case the output of CUTSDP produces as default output with `PPAR = 1`:

```
EQUICUT bounds based on CUTSDP
Graph is of size n = 60, m = 885
   Initial upper bound =  --

CUTSDP called with:
   number of large adds with    60 inequalities:    3
   number of small adds with    18 inequalities:    2
   before small add:  3 extra iterations
SDP solved with:
   duality gap: 1.0e-05
   feasibility: 1.0e-06

  0 ineq.,                    ubd =   --  , lbd = 354.46, cpu =     0.88
 60 ineq.,  60 new,   0 rem, ubd =   --  , lbd = 357.71, cpu =     1.90
 78 ineq.,  18 new,          ubd =   --  , lbd = 359.70, cpu =     2.87
 96 ineq.,  18 new,          ubd =   --  , lbd = 360.64, cpu =     4.68
150 ineq.,  60 new,   6 rem, ubd =   --  , lbd = 360.64, cpu =     6.60
168 ineq.,  18 new,          ubd =   --  , lbd = 361.82, cpu =     8.13
186 ineq.,  18 new,          ubd =   --  , lbd = 362.43, cpu =    10.78
244 ineq.,  60 new,   2 rem, ubd =   --  , lbd = 362.43, cpu =    14.00
262 ineq.,  18 new,          ubd =   --  , lbd = 363.03, cpu =    16.62
280 ineq.,  18 new,          ubd =   --  , lbd = 363.51, cpu =    21.58

CUTSDP completed
Cpu time =    21.58
Termination Status: 0

Output written into 'output.m'

Lower bound for equicut: 363.51
```

From the command line, some default parameters can be changed by using the following options (see also Section 5.1).

`-D`	with values 0 or 1 for `dat_int`
`-P`	with values 0, 1 or 2 for `ppar`
`-S`	with values 0 or 1 for `feasol`
`-U`	with a valid upper bound for `ubd`
`-V`	with values 0 or 1 for `proj_e`
`-Z`	with values 0 or 1 for `zeroone`

As example, the call '`% bisd graphs/ex60a 20 0 0 -P 2 -S 1`' computes the value of the basic relaxation of the bisection of graph `ex60a` into 40 and 20 vertices, provides full output, and also constructs a feasible solution:

```
BISECTION bounds based on SDP
   Graph is of size n = 60, m = 885
   Bisection into 40 and 20 vertices
   Initial upper bound =  --

CUTSDP called with:
   number of large adds with    60 inequalities:    0
   number of small adds with    18 inequalities:    0
   before small add:  3 extra iterations
SDP solved with:
   duality gap: 1.0e-05
   feasibility: 1.0e-06

   0 ineq.,                   ......................................

it    alp    ald    rgap      prim       dual     int  pfeas   dfeas
 0   1.00   1.00  1.2e+00   400.0000   -60.0000    1  2.3e-13 0.0e+00
 1   1.00   1.00  5.9e-01   392.8943   162.8943    1  8.0e-13 9.9e-16
 2   1.00   0.46  3.4e-01   379.2637   249.3493    1  5.7e-13 4.0e-15
 3   1.00   0.41  2.1e-01   358.4192   282.6596    1  2.3e-12 3.5e-15
 4   0.58   0.37  1.4e-01   344.7880   295.0344    1  1.7e-12 3.4e-15
 5   0.52   0.34  9.9e-02   332.2687   299.4569    1  2.6e-12 3.1e-15
 6   0.58   0.47  6.0e-02   322.0726   302.8231    1  6.3e-13 3.7e-15
 7   0.66   0.53  3.2e-02   314.6158   304.4556    1  1.2e-11 4.2e-15
 8   1.00   0.60  9.7e-03   308.0785   305.1030    1  5.0e-11 3.5e-15
 9   1.00   0.67  2.9e-03   306.2787   305.3934    1  6.5e-11 2.3e-15
10   1.00   0.72  8.5e-04   305.7490   305.4901    1  1.5e-11 2.9e-15
11   1.00   0.72  2.5e-04   305.6013   305.5236    1  2.9e-10 2.7e-15
12   1.00   1.00  6.4e-05   305.5593   305.5398    1  1.6e-09 2.9e-15
13   0.37   1.00  4.1e-05   305.5534   305.5410    1  5.7e-08 3.2e-15
14   0.52   1.00  2.3e-05   305.5484   305.5412    1  7.4e-08 2.3e-15
15   1.00   1.00  5.8e-06   305.5431   305.5413    1  8.8e-09 2.1e-15

ubd = 331.00, lbd = 305.54, cpu =     0.87

CUTSDP completed
Cpu time =     0.87
Termination Status: 0

Output written into 'output.m'
```

```
Lower bound for bisection: 305.54
Upper bound for bisection: 331.00
Relative gap (in percent):   7.69
```

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