An SDP-Package for SCIP

Sonja Mars* Lars Schewe

17th August 2012

We present a SCIP-plug-in for solving general mixed-integer semidefinite programmes. We use an interior-point algorithm for solving the SDP-relaxations in every node and the branching framework of SCIP. Additionally we implemented an approximation scheme using cuts generated with the eigenvectors of the current node solution. Both can be used for general SDPs, no structure of special problems like max-cut is used. Combining both methods produces promising results. As examples we solve problems arising from Truss Topology Design with discrete bar areas and some max-cut instances.

1 Problem formulation

Many applications and combinatorial optimization problems can be formulated as semidefinite programme with integer or binary variables. We call the following problem a mixed-integer semidefinte programme:

$$\begin{aligned} \min\ & c^T x \\ \text{s.t.}\ & \textstyle\sum_{i=1}^{n} A_i x_i - A_0 = Y \\ & Y \succeq 0 \\ & x \in \mathbb{Z}^d \times \mathbb{R}^{n-d}. \end{aligned} \tag{MISDP}$$

The A_i are $m \times m$-dimensional, symmetric matrices and can be in blockdiagonal structure. The objective coefficients are $c \in \mathbb{R}^n$. Relaxing $x_i \in \mathbb{Z}$ we obtain a standard SDP which can be solved using any SDP-Solver.

There are two main sources for such problems: Semidefinite programs model a wide variety of applications (especially with an engineering background). Considering additional discrete decisions then leads to a mixed-integer semidefinite program. Or we can obtain such a program from explicitly reintroducing binary variables in the semidefinite relaxation of combinatorial optimization problems. In Section 4 examples of both types are considered.

*Supported through CRC 805, funded through the German Science Foundation (DFG).

The approach presented here does not make any further structural assumptions on MISDP. We, specifically, do *not* assume that the trace of Y is constant. The drawback of this flexibility is, of course, that the code is not competitive with special-purpose codes for specific problems.

The core of the approach taken here is to start with a simple branch-and-cut approach using SDP instead of LP relaxations. To compute cheaper lower bounds we also use certain LPs that are augmented with outer-approximation cuts induced by the underlying SDP cone.

2 Installation

Our SDP-package only works with SCIP-version 3.0 or higher, because some of the relaxator features were not implemented in older versions. We provide a Makefile, which can be used for compiling our code and linking it against SCIP and DSDP. A detailed description how to do that can be found in the file `install.txt`. We also prepared a doxygen documentation which can be built with the command `make doc`. Afterwards it can be found in the doc directory. After building you can use our code calling `./main instancefile.dat-s` or `./main instancefile.dat-s parametfile.set`. Using a parameter file you can decide which algorithm is used. The default settings do a pure branch and bound with SDPs solved at every node. Changing the scip-parameters for the frequencies of LP-solver and SDP-relaxator will change the solving procedure. If the LP-solver frequency is set to -1, it is never called and the problem will be solved using only the SDP-relaxator. It is also possible to set the frequency of the SDP-relaxator to -1, if this is the case, the problem will be solved approximately using the eigenvalue-cuts described in section 3.1. Additionally you can change every parameter, you can change in scip. A detailed description can be found at `http://scip.zib.de/`.

3 Implementation details

In this section we will give a brief overview about the implementation details of our code. As our package is an extension of SCIP [Ach09], [sci12], we will use the SCIP terminology for the different components. We provide a constraint handler for SDP constraints, a reader for sdpa files and a relaxator for computing SDP relaxations. Additionally, we provide a general interface for SDP solvers that allows to use different solvers to used in the relaxator. At the time of writing, the only implemented interface is to DSDP [BY08]. Interfaces to other solvers is work in progress.

We provide two main choices for computing relaxations: Either by solving the SDP relaxation using the above-mentioned interface or by computing an LP-based relaxation using an outer approximation of the SDP cone. The frequency with which these relaxations are computed can be set via the standard SCIP parameter interface. The default setting is to only use the SDP relaxation and never to compute an LP-based relaxation (described in Section 3.1).

Data structures and Reader

To input MISDPs in the form of MISDP, we provide a reader for files in the standard sdpa format and a minor extension of this format to specify integrality restrictions (cf. `data-format.txt` in the distribution). We assume that the matrices in MISDP have a common block-diagonal structure.

$$\sum_{i=0}^{n} \begin{pmatrix} A_{i0} & & & \\ & A_{i1} & & \\ & & \ddots & \\ & & & A_{ik} \end{pmatrix} - \begin{pmatrix} A_{00} & & & \\ & A_{01} & & \\ & & \ddots & \\ & & & A_{0k} \end{pmatrix} = \begin{pmatrix} Y_0 & & & \\ & Y_1 & & \\ & & \ddots & \\ & & & Y_k \end{pmatrix}$$

and

$$\begin{pmatrix} Y_0 & & & \\ & Y_1 & & \\ & & \ddots & \\ & & & Y_k \end{pmatrix} \succeq 0$$

For each of these blocks we add an SDP constraint of the following form to the constraint pool of SCIP:

$$\sum_{i=1}^{n} B_{ij} x_i \succeq B_{0j}$$

Standard linear constraints are handled by the appropriate constraint handlers of SCIP. Internally, the matrices of the constraint are stored in a compressed sparse format and all the data necessary is kept in a class called `SdpCone`. Conversion to this format is handled by the constraint handler. As the sdpa format allows to directly specify a block-diagonal decomposition, the split in different constraints can be handled by the reader. In the current version not much work is done to detect linear constraints that are not marked as such in the original instance.

Preprocessing

The SDP constraints cannot be given unmodified to an SDP solver as the solver does not, by itself, take care that standard constraint qualifications are fulfilled. For our purposes, this means that SDP solvers are, in general, less forgiving with regard to the concrete form in which the problem is given. We have to, for instance, explicitly remove fixed variables and redundant constraints. During branch-and-bound it is a common occurence that variables get fixed. This makes it necessary to perform this type of preprocessing often and efficiently. The interface to the preprocessed `SdpCone` is provided by STL-type iterators. This will allow us, in future, to make modifications to the handling of fixed variables with minimal or no changes in the solver interface.

Constraint-handler

Our semidefinite constraint-handler understands the `SdpCone` class described above, and uses it for presolving and constraint checking. The most important methods of the constraint-handler is indeed the feasiblity checking routine. The feasiblity of an SDP constraint can be checked by ensuring that the smallest eigenvalue of the resulting matrix is nonnegative or equivalently that the Cholesky factorization exists. Even though the latter alternative is, in general, faster, we use an explicit eigenvalue check as we often reuse the resulting minimal eigenvector in the so-called enforcing methods. For the computations we use routines from `LAPACK` [ABB+99]. Additionally, we provide enforcing methods, one for LP-solutions and one for pseudo-solutions. We explain the enforcing method in Section 3.1 in detail.

Relaxator

The SDP-solving process is done in the so-called relaxator which uses the interface to the SDP solver. Every time the relaxator is called, we need to get the cleaned-up data out of the `SdpCone` and the LP to hand it over to the SDP-solver. As the interior-point-methods used by targeted solvers are not able to do warmstarts, the loss in performance incurred through the explicit copying is neglible. We do not call the SDP-solver if there are no more variables left, i.e. all variables are fixed. In this case we only check feasibility.

After the solver is run, the SDP solver returns its status in two different variables: It returns a *convergence status*: *converged* or *not converged*. It also returns a *result status*: *feasible*, *infeasible*, *unbounded*, or *unknown*.

We now need to distinguish the following cases:

- SDP-solver converged
  - result is feasible or unknown
    ⇒ follow solution-handling-procedure (see below)
  - result is infeasible
    ⇒ cut off node
  - result is unbounded
    ⇒ set bound to infinity and go on with branch and bound process
- SDP-solver **not** converged
  - result is unknown
    ⇒ use a penalty function to determine whether there is a feasible point or not (if all integer variables are fixed, follow solution-handling-procedure (see below))
  - result is infeasible or unbounded
    ⇒ return `didnotrun` (if all integer variables are fixed cut the node off)
  - result is feasible

$\Rightarrow$ follow solution-handling-procedure (see below)

After every successful solving, in the cases described above, the following procedure is applied.

Solution-handling-procedure:

(i) Hand over the objective value to SCIP (if we are not converged we hand over the objective value of the parent node).

(ii) Take the solution and check the bounds.

- If the bound-check was successful:
  - (iii) separate the solution and
  - (iv) choose branching candidates.
- If the bound-check failed, but the solution was feasible and converged:
  - (iii) use a penalty function to determine whether there is a feasible point or not.

In certain cases, we need to make sure that our problem is still feasible. In these cases, we solve a modified auxiliary problem: For this we relax all constraints and introduce a new variable that measures the constraint violation. We then try to minimize the constraint violation to obtain a feasible point. As DSDP has a built-in method for this purpose the current interface directly calls this method. Detailed information about the implementation in DSDP can be found in the DSDP User Manual [BY05].

3.1 A coarser relaxation

As already mentioned in Section 3, we sometimes use a second type of relaxation for MISDP instead of solving the SDP relaxation.

To this end we compute an LP relaxation of the problem as follows:

We solve a LP consisting of all the linear constraints and variables of the MISDP. This solution typically violates one or all of the semidefinite constraints. This means that there exists a negative eigenvalue with eigenvector $\bar{v}$, i.e.

$$\bar{v}^T(\sum_i A_i\bar{x}_i - A_0)\bar{v} < 0$$

for some solution $\bar{x}$. Since being positive semidefinite means that $v^T(\sum_i A_i x_i - A_0)v \geq 0$ for all v, we enforce the LP by adding the cut

$$\begin{aligned} \bar{v}^T(\sum_i A_i x_i - A_0)\bar{v} &\geq 0 \\ \Leftrightarrow \sum_i \bar{v}^T A_i \bar{v} x_i - \bar{v}^T A_0 \bar{v} &\geq 0 \end{aligned}$$

This cuts off the current solution. The augmented LP is then resolved until we find a feasible solution or we make not enough progress in the objective function. In the latter case, we start branching and add additional cuts.

The main advantage of this approach is not the quality of the bound that it computes, but its speed on certain problem classes. The bound computed by the solution of the full SDP is at least as strong as the bound computed by this approach. For certain problems, however, the number of iterations needed is quite small. It allows us then to faster enumerate nodes deeper in the tree and, thus, find feasible solutions quickly. In our tests we observed a favorable behavior on the truss instances and a strongly negative behavior on the MAXCUT instances.

4 Results

We consider two types of problems for our computational experiments: Truss topology optimization problems and MAXCUT problems.

The truss topology optimization problem as considered here is at its core a nonlinear optimization problem. We need to design a truss in such a way that the so-called compliance of the truss with respect to an outside force is minimized. In the classical cases the design variables, which correspond to the cross-sectional area or the volume, of the bars are be continuous. A typical extension, however, is consider only discrete areas, which can be modeled in the form of MISDP.

The MAXCUT problem is representative for a classical source of SDPs. A combinatorial optimization problem is formulated as a nonconvex 0/1- (resp. ± 1)-quadratic program and from that a semidefinite relaxation is derived. If we retain the integrality conditions on the variables we also obtain a problem in the form of MISDP.

For both of these problems specialized approaches exist that outperform the general approach presented here. The examples here are mainly presented to give an impression of the flexiblity of the package.

4.1 Truss Topology Design

The aim is to find an optimal topology for a truss from a given groundstructure consisting of bars and nodes under a volume constraint, so that the truss can carry the external load. Optimal in this context means that we want to minimize the energy stored in the truss. This leads to the following non-convex non-linear formulation:

$$
\begin{aligned}
\min_{u} \ & \tfrac{1}{2} f^T u \\
\text{s.t.} \ & \textstyle\sum_e A_e x_e u = f \\
& \textstyle\sum_e l_e x_e \leq V_{\max} \\
& u \in \mathbb{R}^d \\
& x \in \mathbb{R}^n \\
& x_e \geq 0, \qquad e = 1, \ldots, n
\end{aligned}
$$

where x_e is the bar area of bar e and u is the displacement of a node. As described in [BTN01], this form is equivalent to the following semidefinite programme:

$$\begin{aligned} \min\ & \tau \\ \text{s.t.}\ & \begin{pmatrix} 2\tau & f^T \\ f & \sum_e A_e x_e \end{pmatrix} \succeq 0 \\ & \sum_e l_e x_e \leq V_{\max} \\ & \tau \leq \tau_{\max} \\ & x_E, \tau \in \mathbb{R} \\ & x_e \geq 0, \qquad e = 1, \ldots, n. \end{aligned} \tag{SDP}$$

As we cannot expect that every bar area can x_e be produced, one extension of this problem is to take only bar areas from a finite set D. Therefore we now use integer variables $x_e \in D = \{0, 1, \ldots, D_{\max}\}$. So we obtain a Mixed-Integer SDP, which can be solved using our package.

Figure 1 – Two exemplary cantilever instances with twelve nodes, two different loads and 47 potential bars. In the left picture (m2-d2-17) the maximal volume was 17, in the right one, we allowed an overall volume of 25 (m3-d2-25).

Figure 2 – Examples for bridges with twelve nodes, three different forces and 49 potential bars. In the left picture we only had to decide if a bar exists or not (b1-d1-20). On the right side, there are two different bar areas allowed (b3-d2-30).

The bridge of figure 3 took about four days of computation. The other instances took between five and fifty minutes.

In table 1 we can observe that the relaxations of MISDPs for trusses yield good lower bounds. The root relaxation is already close to the optimal solution in most of the

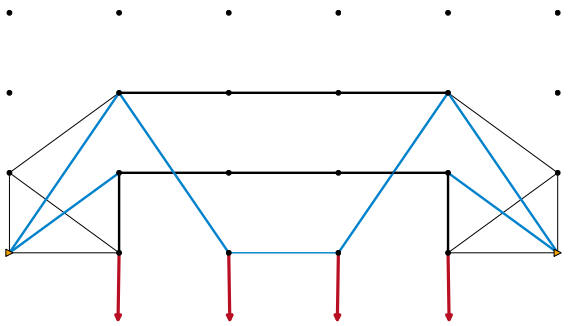

Figure 3 – A bigger example with some special bars (blue). Here we had 24 nodes with 188 bars allowed and two different bar areas.

					best sol		first sol		
instance	vars	cons	time	nodes	value	node	node	gap	root gap
b1-d1-20	50	8	7.28	139	1.68131	139	139	0.0%	3.72%
b3-d2-30	99	57	839.94	8141	0.9876	8141	922	19.98%	10.817%
m2-d2-17	95	53	46.85	481	0.3809	481	379	598.43%	7.24%
m3-d2-25	95	53	283.94	2676	0.2896	2676	1772	186.05%	11.55%
b1-a3-d1-20	148	154	20.89	146	1.44496	140	140	0.0%	2.67%
b2-a3-d2-20	197	203	323.47	1379	1.09532	1313	1313	0.0%	7.09%
b3-a3-d2-30	197	203	391.38	1673	0.80367	1591	1591	0.0%	4.93%
b-big	82	10	178.39	1191	2.54641	1189	1147	0.12%	4.76%
m1-a3-d1-20	142	148	55.77	394	0.41424	384	384	0.0%	7.09%
m2-a3-d2-20	189	195	310.4	1212	0.33554	1199	677	757.64%	15.86%
m3-a3-d2-30	189	195	777.45	2963	0.23078	2963	2875	201.06%	10.52%
m4-a4-d1-20	142	148	51.51	364	0.39382	360	360	0.0%	6.81%
m5-a4-d2-20	189	195	397.27	1561	0.32693	1539	1503	455.51%	17.45%
m6-a4-d2-30	189	195	1014.74	3997	0.22206	3997	2574	259.59%	11.12%

Table 1 – Some examples from Truss Topology Design

problems. It is, however, very difficult to find feasible solutions, most of them are found at the leafs of the branch and bound tree.

4.2 Maximum Cut

One standard example where SDP-relaxations are used in combinatorial optimization problems is the maximum cut problem. Given an edge-weighted graph $G = (V, E)$ with vertices $V = \{1, \ldots, n\}$, edges $ij \in E$, where i and j are the endpoints of edge ij and weights c_{ij}, with $c_{ij} = 0$ for $ij \notin E$ and $c_{ii} = 0$.

The *cut* $\delta(S)$, with $S \subseteq V$, is defined as all edges ij with $i \in S$ and $j \in V \setminus S$. Now we want to find a set S, such that the sum of all weights of the edges in the cut is maximal, i.e.

$$\max_{S \subset V} \sum_{ij \in \delta(S)} c_{ij}. \tag{1}$$

Now we are going to model this problem using variables $x_i \in \{-1, 1\}$ for every vertex i. Then (1) can be written as

$$\begin{array}{ll} \max & \sum_{i<j} c_{ij} \frac{1-x_i x_j}{2} \\ \text{s.t.} & x_i \in \{-1, 1\}, \quad i = 1, \ldots, n. \end{array}$$

The product of two variables $x_i x_j$ is positive if the vertices i and j belong to the same set, i.e. $i, j \in S$ or $i, j \in V \setminus S$. It is negative if the corresponding edge ij is in the cut, so the two edges belong to different sets.

This problem can be formulated as Semidefinite Programm (see e.g. [Hel00]). Therefore we need to transform the objective function in the following way:

$$\sum_{i<j} c_{ij} \frac{1 - x_i x_j}{2} = \frac{1}{4} \sum_{i=1}^{n} \left(\sum_{j=1}^{n} c_{ij} x_i x_i - \sum_{j=1}^{n} c_{ij} x_i x_j \right) = \frac{1}{4} x^T (\text{Diag}(Ce) - C) x),$$

where $\text{Diag}(Ce)$ maps the vector Ce to a diagonal matrix in an canonical way. For $L(G) = \frac{1}{4}\text{Diag}(Ce) - C$ we can reformulate our problem:

$$\max_{x \in \{-1,1\}^n} x^T L(G) x.$$

For some real $m \times n$-dimensional matrices A, B we define

$$\langle A, B \rangle = tr(B^T A) = \sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij} b_{ij}$$

The semidefinite relaxation is obtained using the facts that $x^T L(G) x = \langle L(G) x, x \rangle = \langle L(G), x x^T \rangle$ and that $x x^T$ is always positive semidefinite for $x \in \{-1, 1\}$. The diagonal entries of $x x^T$ are always equal to one and its rank is also equal to one. Characterizing the matrix $X = x x^T$ using these constraints yields a SDP:

$$\begin{array}{ll} \max & \langle L(G), X \rangle \\ \text{s.t.} & \text{diag}(X) = e \\ & X \succeq 0 \\ & x_{ij} \in \{-1, 1\}. \end{array}$$

For modelling this problem in the standard SDP-form, we need to look at the complement and transform the variables to $\{0, 1\}$, because it is not possible to model $x \in \{-1, 1\}$ in the sdpa-format. Our new variables y_i are equal to zero, whenever x_i is equal to minus one and $y_i = 1$ if and only if $x_i = 1$.

Table 2 shows that the root node relaxation yields a very good lower bound, better than in the case of truss topology problems. Additionally note that the first feasible solution is also found in the root node by a trivial heuristic. But the quality of this first solution is not very good. The quality of the bound at the root often leads to immediate termination if the optimal primal solution is found.

instance	vars	cons	gap	time	nodes	best sol node	first sol gap	root gap
maxcut30-0	435	1	0.0%	981.38	788	788	161.27%	4.48
maxcut30-1	435	1	0.0%	101.48	73	73	193.36%	2.86
maxcut30-2	435	1	0.0%	123.59	88	88	173.75%	1.85
maxcut30-3	435	1	0.0%	200.83	151	151	184.15%	4.48
maxcut30-4	435	1	0.0%	169.77	126	126	170.21%	4.61
maxcut40-0	780	1	0.0%	11753.02	1975	1975	151.41%	4.95
maxcut40-1	780	1	0.0%	6116.32	1073	1073	162.38%	4.22
maxcut40-2	780	1	0.0%	10152.97	1632	1632	161.42%	5.12
maxcut40-3	780	1	0.0%	25177.06	4206	4206	156.24%	4.57
maxcut40-4	780	1	0.0%	3939.45	621	621	164.0%	6.26
maxcut40-5	780	1	0.0%	2641.39	413	413	163.42%	5.05
maxcut40-6	780	1	0.0%	5012.19	813	813	166.95%	5.58
maxcut40-7	780	1	0.0%	21165.88	3205	3202	157.43%	5.36
maxcut40-8	780	1	0.0%	2082.6	358	358	159.48%	3.43
maxcut40-9	780	1	0.0%	3641.79	663	663	163.8%	6.29
maxcut50-0	1225	1	0.0%	119632.73	5719	5719	158.66%	4.57
maxcut50-1	1225	1	0.0%	144649.92	7438	7438	152.63%	4.02
maxcut50-2	1225	1	0.0%	18994.19	1033	1033	155.6%	3.46
maxcut50-3	1225	1	0.0%	12178.69	650	650	161.79%	3.52
maxcut50-4	1225	1	0.0%	109428.59	6356	6356	144.87%	3.58
maxcut50-5	1225	1	0.0%	253946.49	14562	14454	150.42%	3.83

Table 2 – Results for max-cut instances

5 Conclusion

The package presented here is able to solve general mixed-integer semidefinite programs. The core ingredients of the solver are directly using an SDP solver in a branch-and-cut framework or to use an LP relaxation of the underlying problem. These two approaches can also be combined. We showed preliminary results for two different problem types: Truss topology design and MAXCUT. Even though, we are not competitve with special purpose software for these problems, we can solve small instances to optimality in reasonable time.

Later versions will include interfaces to other SDP solvers and improved presolving routines.

Acknowledgement

The authors like to thank Alexander Martin, Marc Pfetsch, Jakob Schelbert, Michael Stingl and the SCIP-Team for a lot of discussions and assistance concerning software and problem formulations. Jakob Schelbert wrote the main part of the sdpa-reader. Especially Marc Pfetsch and the SCIP-Team have been very helpful and supported a lot of stuff directly in the SCIP-code. Additionally, thanks to the Collaborative Research Center 805 for giving interesting insights into mechanical engineering. We also thank Anne Philipp for testing and reporting bugs.

References

[ABB+99] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.

[Ach09] Tobias Achterberg. Scip: Solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009. `http://mpc.zib.de/index.php/MPC/article/view/4`.

[BTN01] Aharon Ben-Tal and Arkadi Nemirovski. *Lectures on Modern Convex Optimization*. Society for Industrial and Applied Mathematics, Philadephia, PA, 2001.

[BY05] Steven J. Benson and Yinyu Ye. DSDP5 user guide – software for semidefinite programming. Technical Report ANL/MCS-TM-277, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, September 2005. `http://www.mcs.anl.gov/~benson/dsdp`.

[BY08] Steven J. Benson and Yinyu Ye. Algorithm 875: DSDP5 – software for semidefinite programming. *ACM Trans. Math. Softw.*, 34(3), 2008.

[Hel00] Christoph Helmberg. *Semidefinite Programming for Combinatorial Optimization*. 2000. Habilitationsschrift.

[sci12] Scip 3.0, 2012. `http://scip.zib.de`.

Sonja Mars
Fachbereich Mathematik
TU Darmstadt
`smars@mathematik.tu-darmstadt.de`

Lars Schewe
Dept. of Mathematics
Friedrich-Alexander-Universität Erlangen-Nürnberg
`lars.schewe@math.uni-erlangen.de`