Documentation of a lift-and-project cut generation module for COIN-OR

October 20, 2006

This document briefly presents an implementation for `COIN-OR` of the lift-and-project procedure executed in the LP simplex tableau as described by Balas and Perregaard [5]. This lift-and-project cut generator is available under the Common Public License [10] (therefore it is an open-source code). The implementation follows the standards of the COIN-OR Cut Generation Library: `Cgl` [7].

This document does not cover the algorithmic details of the procedure and its implementation, but its intention is just to provide a reference for the usage of the code. For an in-depth description of the algorithm and of the mathematical concepts on which it is based, the reader is invited to refer to the paper of Balas and Perregaard [5]; for detailed descriptions of implementations of the procedure and extended computational experiments, the reader is invited to refer to Michael Perregaard Thesis [11] and the working paper [**?**].

Along with the procedure described in [5], the procedure implements two closely related variants [4, 6] which can be used as options.

Consider a mixed integer linear program of the form:

$$\begin{aligned}
&\min c^T x \\
&\text{such that:} \\
&Ax \geq b, \qquad\qquad\qquad\qquad (MILP) \\
&l \leq x \leq u, \\
&x_j \in \mathbb{Z}\ j \in I, \quad x \in \mathbb{R}^n
\end{aligned}$$

where $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ and$l$ and $u \in \mathbb{R}^n$ are the (possibly infinite) lower and upper bounds on the variables and finally $I \subseteq \{1, \ldots, n\}$ is the subset of integer constrained variables.

We denote by (LP) the continuous relaxation of $(MILP)$.

The lift-and-project procedure generates a set of linear inequalities cutting the optimal solution of (LP) x^*. More precisely a *round* of lift-and-project generates a cut for each of the integer constrained variables $x_k \in I$ whose value x_k^* is not integral. To obtain each cut, the method proceeds by performing pivots in the simplex tableau of (LP). These pivots have the property that each of them makes the *Intersection Cut* obtained from the row corresponding to variable x_k more violated in the next tableau (obtained after the pivot) than in the current one.

The procedure stops either when the current intersection cut is optimal in the sense that it is the most violated by x^* which can be obtained by taking into account only the integrality of variable x_k or when a prescribed pivot or time limit is attained.

When the optimal lift-and-project cut is found it is usually `strengthened` in a cut which can be shown to be equivalent to the *Gomory* cut in the current basis. It is important to note that the optimal basis from which the strengthened lift-and-project cut is generated is often an infeasible basis and also that the cut obtained is different (and usually stronger) than the Gomory cut from the optimal tableau of (LP).

1 Usage and parameters of the procedure

Like all generator from the Cut Generation Library `Cgl`[7], the class `CglLandP` which implements the lift-and-project cut generation procedure inherits from the class `CglCutGenerator` and its main function is `CglLandP::generateCuts` which performs a round of lift-and-project cuts at the current optimal basic solution.

To generate each individual cut the function `CglLandP::generatesCuts` uses the class `CglLandPSimplex` which implements the lift-and-project pivoting procedure briefly described above.

In this procedure, all the tableau operations (such as computing rows, columns and pivoting) are performed by using the standard functions provided in the LP solver at hand through the use of an `OsiSolverInterface`. Therefore, any `OsiSolverInterface` which provides an implementation of the functions

- `pivot`,
- `getBInvARow`,
- `getBInvACol`,

(see documentation in `OsiSolverInterface`) can be used for generating the lift-and-project cuts.

Unfortunately at the moment only the interface for `Clp` [8] provides these 3 functions.

For the generation of the cuts besides the classical procedure (which is used by default by the generator), two variants can be optionally used:

- a procedure which uses an alternative criterion for selecting pivoting variables based on searching for the adjacent basis which has the most violated intersection cut [6].
- a procedure which after each pivot replaced the source row by its modularized version [**?**],

The cuts can either be separated in the full space or in the reduced space (as described in [2]), the default procedure is to separate the cuts in the reduced space.

The cut generation procedure has a certain number of limits which can be changed by the user. The main one is the limit on the number of cuts generated at each round which is set to 50 by default (the behavior is to generate a cut for each of the 50 most fractional variables in the solution to cut x^*.

The separation of each cut is also limited by several parameters. In particular there is an upper limit on the number of pivots performed for separating each cut, which is 20 by default.

Once a cut has been separated, a number of operations are performed in order to *clean* its small coefficients and possibly reject it following a number of criterion explained in Appendix A.

1.1 Summary of the parameters and their default values

Below we give the name the default value and a brief description of each of the parameters. These parameters can be changed by using the member function `parameters()` of the class `CglLandP`.

1.1.1 algorithm choices

- `selectionRule` Rule for choosing pivoting variables. Possible values are:
  - `mostNegativeRc` Select variable with most negative reduced cost for entering the basis,
  - `bestPivot` Select leaving and entering variables according to the Most Violated Cut Selection Rule.

  The default value is `mostNegativeRc`.
- `modularize` If set to 1, after each pivot modularize source row (set to 0 by default)

- `reducedSpace` If set to 1, perform the cut generation in the reduced space (set to 1 by default).

1.1.2 Per round limits

- `maxCutPerRound` Maximum number of cuts generated in each round, the default value is 50.
- `timeLimit` global time limit in seconds for performing one round of separation (default value is $+\infty$).
- `away` In the current LP optimum, a basic variable have to be at least this much away from an integer value for a cut to be generated for the corresponding row. The default value is 5.10^{-3}.

1.1.3 Limits for the separation of each cut

- `pivotLimit` Maximum number of pivots performed for each cut. The default value is 20.
- `degeneratePivotLimit` limit the number of consecutive degenerate pivot, default value is 0.
- `singleCutTimeLimit` time limit in seconds for each cut separation (default value is $+\infty$).

1.1.4 Cut parameters

- `scaleCuts` If set to true cuts are scaled to have +-1 right-hand-side. Most linear programming solvers (`Clp`, `Cplex`, `Xpress`,...) already include an automatic scaling of the problem and therefore by default no scaling is performed.

1.1.5 Pivoting tolerances

- `pivotTol` if the pivot element $\overline{a}_{ij}$ is less than this value we don't consider this pivot, the default tolerance is 10^{-6}.

2 Small example

We give here a small example where one round of lift-and-project cut is performed on the problem stored in the `OsiSolverInterface si`. The default parameters for the time limit and the maximum number of pivots per cut are changed respectively to 10 seconds and 2 pivots.

```
CglLandP landpGen;

landpGen.parameter().timeLimit = 10.;
landpGen.parameter().pivotLimit = 2;

OsiCuts cuts;

landpGen.generateCuts(si, cuts);

si.applyCuts(si);
```

References

[1] E. Balas. Disjunctive programming : Properties of the convex hull of feasible points. *MSRR 348,Carnegie Mellon University*, 1974. Also published in *Discrete Applied Mathematics*, 89, 1998, 3-44.

[2] E. Balas, S. Ceria, and G. Cornuéjols. A lift-and-project cutting plane algorithm for mixed 0-1 programs. *Mathematical Programming*, 58:295–324, 1993.

[3] E. Balas and R. G. Jeroslow. Strengthening cuts for mixed integer programs. *European J. Oper. Res.*, 4(4):224–234, 1980.

[4] E. Balas. Generating Deepest Mixed Integer Cuts by Iterative Disjunctive Modularization. Research Report. May 2002.

[5] E. Balas and M. Perregaard. A precise correspondence between lift-and-project cuts, simple disjunctive cuts, and mixed integer Gomory cuts for 0-1 programming. *Math. Program.*, 94(2-3, Ser. B):221–245, 2003. The Aussois 2000 Workshop in Combinatorial Optimization.

[6] E. Balas and P. Bonami. Implementation of lift-and-project method in COIN-OR. Working Paper. 2006.

[7] Cgl (Cut Generation Library). *http://projects.coin-or.org/Cgl.*

[8] Clp (COIN-OR Linear Programming) *http://projects.coin-or.org/Clp.*

[9] COIN-OR. *http://www.coin-or.org.*

[10] Common Public License. *http://www.opensource.org/licenses/cpl1.0.php*

[11] M. Perregaard. Generating Disjunctive Cuts for Mixed Integer Programms Ph.D. Thesis, Carnegie Mellon University, 2003.

[12] M. Perregaard. A practical implementation of lift-and-project cuts. International Symposium on Mathematical Programming, Copenhagen (2003).

A Cut checks, and rules for cut rejection

Here we give the details of the procedure applied to validate a cut after its generation and before it is eventually added to the formulation of the problem.

The goal of this procedure is to try and avoid generating invalid cuts, or cuts which intrinsically have a bad numerical behavior and would give trouble to the LP solver. All the rules applied here are simple heuristics and do not give any guarantee that these goals are completely attained. Although most cut generators (for instance the generators available in the `COIN-OR` library `Cgl`) and mixed integer programming solvers implements comparable rules, they are rarely discussed. Since they can play an important role in the quality of the cut generated and the overall cut generation procedure and its integration in a global branch-and-cut, we believe it is worth giving a precise account of the rules we apply here.

Let $\alpha^T x \geq \beta$ be the cut generated by the lift-and-project procedure put in the structural space of the problem. Let

$$\nu = \frac{\max\limits_{i \in \{1,\ldots,n\}} |\alpha_i|}{\min\limits_{i \in \{1,\ldots,n\}:\alpha_i \neq 0} \alpha_i}$$

be the ratio between the coefficient of largest magnitude and the non-zero coefficient of smallest magnitude in the cut. In general we want to avoid having ν to big since it would cause trouble in floating point computation, therefore the first part of the validation procedure is to try and control the size of ν by removing small coefficients in the cut without making the cut invalid. Assume that the cut is scaled so that $\max\limits_{i \in \{1,\ldots,n\}} |\alpha_i| = 1$ and that $\overline{\nu}$ is the maximal value we accept for ν. We try to remove all the coefficients of the cut such that $|\alpha_i| < \frac{1}{\overline{\nu}}$. If $\alpha_i > 0$ and the variable x_i has a finite lower bound l_i, the coefficient is set to 0 and the right hand side β is modified to $\beta - \alpha_i l_i$. Conversely if $\alpha_i < 0$ and the variable x_i has a finite upper bound, the right hand side is modified to $\beta - \alpha_i u_i$. If either the coefficient $\alpha_i > 0$ and the variable x_i has no lower bound or the coefficient is negative and the variable has no upper bound, the coefficient can not be safely removed and the cut is discarded.

If all the coefficient could be safely removed so that in the resulting cut $\nu \leq \overline{\nu}$, we check for the density of the cut obtained:

$$d = \frac{|\{i \in \{1, \ldots, n\} : \alpha_i \neq 0\}|}{n}.$$

If d is bigger than a prescribed value $\overline{d}$ the cut is rejected otherwise we recompute the cut violation and it is sufficiently large the cut is accepted. The value of $\overline{d}$ is usually set as a function of the density of A, the default is $\overline{d} = 1$.