

First Steps with COIN-OR

François Margot¹

January 2009

3. Split Cuts and Branch-and-Cut

Contents

1	Overview	1
2	Pre-defined Cut Generators, Other Solvers	2
3	Adding Split Cuts	3

1 Overview

The goal is to study some simple elements of the branch-and-cut-and-price code (BCP).

Please refer to the first project (see `proj1.pdf`) for installation steps including the definition of required environment variables and generation of the html documentation.

1. Go to the directory PROJ3.

¹Tepper School of Business, Carnegie Mellon University, Pittsburgh, PA 15213-3890, U.S.A. Email: `fmargot@andrew.cmu.edu` .

The structure of this example is similar to the one described in [1]. Three main directories (PROJ3/TM for tree manager, PROJ3/LP for operations at the node level, and PROJ3/Member for other operations) and all the include files are in the directory PROJ3/include.

The code can be compiled by typing:

```
% make
```

If the compilation fails, you might have to modify some of the entries in PROJ3/Makefile. Please refer to Section 7 of `proj1.pdf` for possible solutions.

The `make` command creates an executable PROJ3/bac. The code can be run from the directory PROJ3 using

```
% ./bac shell.par
```

It currently reads the files `shell.lp` (an integer linear program in LP format) and `shell.par` (parameters for the run), and solves the corresponding ILP using BCP and Clp.

The command `make clean` defined in PROJ3/Makefile is useful to re-compile. It erases all the object files of the example as well as the executable `bac`.

Note that as explained in [1] the code of this example will not run on a parallel machine.

2 Pre-defined Cut Generators, Other Solvers

The code can be compiled with the flags `-DLP`, `-DMPS`, `-DUSER_DATA`, `-DTRACE`, and/or `-DGOMORY` (see PROJ3/Makefile, line starting with `USER_DEFINES =`). Exactly one of the flags `-DLP` or `-DMPS` must be defined, making the code read from the LP file `shell.lp` or the MPS file `shell.mps`. The flag `-DUSER_DATA` must be used (see [1] for its meaning). Using the flag `-DTRACE` generates a detailed output. Adding the flag `-DGOMORY` for compilation makes the code use the pre-defined Gomory cut generator based on the class `CglGomory`¹.

The Gomory cut generation is called from the class `BB_lp`² in the method `generate_cuts_in_lp()`. `BB_lp` is derived from `Bcp_lp_user`³.

To generate the Gomory cuts, an object of class `CglGomory` is defined and then the method `CglGomory::generateCuts()` is called. it returns a

¹`Bcp-1.2.1/buildg/include/coin/CglGomory.hpp` or `DocBcp->CglGomory`.

²`PROJ3/LP/BB_lp.cpp` or `DocBcp->BB_lp`.

³`DocBcp->Bcp_lp_user`.

collection of cuts in an object of class `OsiCuts`. Each of these cuts is then put in the pool of cuts to be added during the next iteration by creating an object of class `BB_cut` (the cut representation used in this application, defined in `PROJ3/include/BB_cuts.hpp` and `PROJ3/Members/BB_cut.cpp`) and pushing it in the vector of cuts `new_cuts`.

To switch to `Cplex` from `Clp`, the key method is

```
BB_lp::initialize_solver_interface() .
```

It is called once at the beginning of the enumeration. Currently, it defines a solver of type `OsiClpSolverInterface`. As before, the switch to `Cplex` is done by replacing each occurrence of `OsiClpSolverInterface` by `OsiCpxSolverInterface` in the code. The only files containing those are files `PROJ3/LP/BB_lp.cpp` and `PROJ3/TM/BB_tm.cpp`.

Do the following:

1. Look at the code of `BB_lp::generate_cuts_in_lp()`. Add code in order to generate knapsack cover cuts using the predefined generator.
2. Modify the code to run it using `Cplex`. Recompile and run.
3. Undo the changes made in steps 1 and 2 before continuing.

3 Adding Split Cuts

Starting either from the code you wrote for the previous project or from the code in the file `PROJ3/LP/split_cuts.cpp`, add the generation of split cuts to the code. The method `generate_split_cuts()` in `split_cuts.cpp` prints the set of nonzero coefficients (and corresponding indices) as well as the upper/lower bounds of the cut obtained using the first fractional structural variable. Modifying `generate_split_cuts()` to collect the generated cuts and pass them back to `BQP` is required, as well as some of the commands to tie the procedure with `BQP` (access the current LP solution, solver, ...). Normally, no change to the method `ray_to_row()` is necessary.

4. Find how to access the current solution \bar{x} of the LP relaxation from the method `BB_lp::generate_cuts_in_lp()`.
5. Find how to get a pointer to the `OsiClpSolverInterface` object used to solve the LP relaxation from `BB_lp::generate_cuts_in_lp()`. (Downcasting is likely to be necessary.)
6. Add code that will generate a split cut for each fractional variable in \bar{x} . Use a tolerance of 0.01 for deciding if a value is integer or not.

References

- [1] F. Margot, "BAC : A BCP based branch-and-cut example", IBM Research Report RC22799 (W0305-064) (2003), revised January 2009.