CBC : COIN-OR Branch-and-Cut

A Short Guide to the CBC Command Line Interface

Prof. Haroldo Gambini Santos

www.decom.ufop.br/haroldo/

Universidade Federal de Ouro Preto

February 2017

Contents

1 . Before we start

This guide is intend to show the basic usage and tuning of `CBC`: the COIN-OR Branch-and-cut standalone executable which is called by the command line.

If you do not have it installed in your computer you grab its sources accessing the project page:

> https://projects.coin-or.org/Cbc/

or, more easily, installing one of the pre-built packages available for your computing platform at:

> http://www.coin-or.org/download/binary/CoinAll/

in a package including several other COIN components.

Although a basic knowledge of Integer Programming is assumed, concepts are briefly explained whenever it is possible.

Credits: a large part of the content of this guide was obtained from the `CBC` advanced command line help which you can access by typing "`verbose 15`" followed by "?"' in the `CBC` interactive mode. Thanks also to the nice folks at cbc@list.coin-or.org.

2 . Quick start

If you installed CBC, you can open a shell[1] and type:

```
cbc air03.lp solve solu sol.txt
```

to load problem air03.lp[2], solve it and save the best solution in a file named `sol.txt`.

An example of a customized CBC execution is given bellow. In this case, the parameter `cuts` receives value `Off` and the parameter `passF` receives value `100` before the beginning of the solution process:

```
cbc air04.lp cuts off passF 100 solve solu sol.txt
```

By calling only CBC without parameters you will enter in interactive mode.

```
Welcome to the CBC MILP Solver
Version: 2.9.8
Build Date: May  6 2016
Revision Number: 2277
CoinSolver takes input from arguments ( - switches to stdin)
Enter ? for list of commands or help
Coin:
```

To solve air04.lp as we did before in interactive mode you can enter the following commands:

```
Coin: import air03.lp
Coin: solve
Coin: solu sol.txt
```

1 Terminal in Linux, Command Prompt in Windows
2 Can be downloaded from http://goo.gl/dvlJnV

3 . Terminal Output

To solve your problem cbc uses a “bag of tricks”. It has to dynamically decide how much processing power will be used in different algorithms/search strategies. To keep you informed of the successes (or failures) of these attempts it continually prints messages containing details about the current search status. Understanding these messages is a key step to pinpoint which are the main difficulties in solving your problem. Once you have this information in your hands you can start to tune cbc so that it will perform better considering the type of problem you are working on.

```
    Welcome to the CBC MILP Solver
    Version: 2.9.8

    command line - cbc air03.lp solve solu sol.txt (default strategy 1)
 1  Continuous objective value is 338864 - 0.05 seconds
    Cgl0003I 0 fixed, 0 tightened bounds, 0 strengthened rows, 4 substitutions
 2  Cgl0004I processed model has 120 rows, 8456 columns (8456 integer) and 71651
    elements
    Cutoff increment increased from 1e-05 to 1.9999
 3  Cbc0038I Pass   1: suminf.    8.33333 (22) obj. 341524 iterations 106
    Cbc0038I Pass   2: suminf.    8.33333 (22) obj. 341524 iterations 3
    Cbc0038I Pass   3: suminf.    8.33333 (22) obj. 341524 iterations 70
    Cbc0038I Pass   4: suminf.    7.20000 (20) obj. 342390 iterations 75
    Cbc0038I Pass   5: suminf.    1.50000 (3) obj. 343697 iterations 45
    Cbc0038I Pass   6: suminf.    1.50000 (3) obj. 343697 iterations 55
                                            ...
    Cbc0038I Pass  12: suminf.    0.00000 (0) obj. 362176 iterations 144
    Cbc0038I Solution found of 362176
                                            ...
```

Message 1 indicates that CBC successfully solved the linear programming relaxation of your problem. The objective value of this solution provides a dual bound (338864), which is an optimistic estimate for the optimal solution cost.

Message 2 informs that pre-processing has finished. Ideally, after that step you will have a smaller problem with a better formulation.

Message 3 and all messages starting with Cbc0038I, indicate that cbc is searching for an initial integer feasible solution using the Feasibility Pump (M. Fischetti, Glover, & Lodi, 2005) method. After twelve passes, it has found a solution with cost 362176. We now have a valid solution for the problem and valid bounds for the optimal solution value: [338864,362176]. The performance of CBC will always depend on how close are these bounds. A special attention should be given to messages indicating the progresses in tightening these bounds, which will be discussed in the next paragraphs.

```
                                            ...
    Cbc0038I Full problem 120 rows 8456 columns, reduced to 0 rows 0 columns
 4  Cbc0012I  Integer  solution  of  340160  found  by  DiveCoefficient  after  14
    iterations and 0 n odes (0.97 seconds)
                                            ...
    Cbc0031I 5 added rows had average density of 390.8
 5  Cbc0013I At root node, 5 cuts changed objective from 338864.25 to 340160 in
    2 passes
```

```
Cbc0014I Cut generator 0 (Probing) - 0 row cuts average 0.0 elements, 160
column cuts (160 active)  in 0.840 seconds - new frequency is 1
Cbc0014I Cut generator 1 (Gomory) - 4 row cuts average 1257.2 elements, 0
column cuts (3 active)  in 0.020 seconds - new frequency is -100
Cbc0014I Cut generator 2 (Knapsack) - 0 row cuts average 0.0 elements, 0
column cuts (0 active)  in 0.020 seconds - new frequency is -100
Cbc0014I Cut generator 3 (Clique) - 10 row cuts average 3.7 elements, 0
column cuts (0 active)  in 0.000 seconds - new frequency is 1
Cbc0014I Cut generator 6 (TwoMirCuts) - 0 row cuts average 0.0 elements, 0
column cuts (0 active)  in 0.030 seconds - new frequency is -100
                                        ...
6 Result - Finished objective 340160 after 0 nodes and 11 iterations - took
4.75 seconds (total time 5.06)
Total time 5.14
```

Message 4 indicates the success of the `DiveCoefficient` heuristic in finding a better feasible solution. We now have tighter bounds: [338864,340160].

Messages starting at 5 contain details of the progress achieved in improving the dual bound by generating a series of cuts removing fractional solutions. In this case 3 cuts suffice to close the gap and produce the best possible dual bound: 340160. We're now sure that the previously found solution is the optimal one. Last messages of this log inform how each one of the active `cbc` cut generators performed. By these messages we can observe that only Clique and Gomory(Gomory, 1958) cuts where useful. Gomory cuts, as usual, produced cuts which are much denser.

Finally, Message 6 announces the end of the search, which took only 5 seconds. In this problem, cbc performed noticeably well: a good feasible solution and a good dual bound were obtained at the root node, so that no further exploration in the branch-and-bound tree was needed.

```
                                        ...
7 Cbc0010I After 200 nodes, 46 on tree, 56212 best solution, best possible
55800.3 (148.79 seconds)
Cbc0016I Integer solution of 56174 found by strong branching after 116654
iterations and 263 nodes (152.34 seconds)
Cbc0038I Full problem 615 rows 7673 columns, reduced to 109 rows 110 columns
8 Cbc0012I Integer solution of 56138 found by RINS after 124176 iterations and
300 nodes (156.76 seconds)
Cbc0038I Full problem 615 rows 7673 columns, reduced to 1 rows 2 columns
Cbc0010I After 300 nodes, 43 on tree, 56138 best solution, best possible
55805.8 (156.94 seconds)
Cbc0038I Full problem 615 rows 7673 columns, reduced to 142 rows 143 columns
Cbc0010I After 400 nodes, 55 on tree, 56138 best solution, best possible
55863.5 (166.19 seconds)
                                        ...
```

Message 7 shows the resolution of a harder problem. For this problem (air04), the search advanced in the Branch-and-bound tree. Frequent messages indicate how many nodes were explored and how many nodes still open and need to be explored. Whenever a new integer solution is found (Message 8), information about its cost and which method found it is printed.

4 . Tunning

To modify cbc default settings, we can specify options before finally call the solve command. A command line can include as many options as you want, in the format:

```
cbc air04.lp ... option1 parameter 1 option2 ... solve solu sol.txt
```

Next subsection will briefly present some of the most inportant cbc options and parameters which can impact the search process. Options are presented in the following way:

ShortName : DataType : DefaultValue : LongName

where ShortName indicates the abbreviation necessary to specify the option; DataType indicates if the parameter is an integer, double or string parameter; default value indicates the value which is automatically selected on cbc startup and finally, LongName indicates the full parameter name.

4.1 Pre-processing and root node

Presolve : String : On : Presolve

Presolve tries to remove redundant constraints and to detect tighter bounds for variables to make the problem simpler.

PreT : Integer : 1e-8 : PreTolerance

Use larger values (e.g. 1e-7) if your problem is feasible but presolve is declaring it infeasible.

PassP : Integer : 5 : PassPresolve

Maximum number of passes for presolve.

Preprocess : String : Sos : Preprocess

Controls the pre-processing routines based on integrality constraints. Integer pre-processing tries to produce a stronger formulation, i.e. one with a better lower bound in case of minimization. This can slow down the start of the search but usually pays off as a smaller branch-and-bound tree is explored if a tighter formulation is obtained.

On : turns on integer pre-processing;

Save : saves the pre-processed problem in presolved.mps;

Equal : transform ≤ cliques into equalities;

Aggregate : tries to create additional variables representing groups of variables;

Sos : creates Special Ordered Sets to improve branching.

Multiple : Integer : 0 : MultipleRootPasses)

Executes the root phase (in parallel if threads are available) and collect all solutions and cuts generated. The format is aabbcc where:

aa : number of extra passes;

bb : non-zero values specify the number of threads, otherwise uses thread settings;

cc : number of times that the root phase is repeated.

RandomC : Integer : -1 : RandomCbcSeed

If you want CBC to give the same results on repeated executions use the same random seed. If you have time you may want to execute CBC with several random seeds and collect the best solution found in those different executions. Use 0 if you want CBC to use the current time.

4.2 Linear Programming Relaxation

The MIP resolution process starts by solving the linear programming relaxation (LP) of the model. The algorithm used to solve this initial model can be selected by including in the command line, just before the *solve* argument one of the following options:

DualSimplex : use the dual simplex method;

PrimalSimplex : uses the primal simplex algorithm;

Barrier : uses the use primal dual predictor corrector algorithm, can be significantly faster n some large models.

4.3 Branch-and-bound

Node : String : Hybrid : NodeStrategy

Strategy for selecting the next unprocessed node.

Fewest : selects node with where fractional values of variables are closest to integer values;

UpFewest : selects node with where fractional values of variables are closest to integer values, branching is performed rounding up first;

DownFewest : selects node with where fractional values of variables are closest to integer values, branching is performed rounding down first;

UpDepth : performs a depth-first-search, rounding up first;

UpDepth : performs a depth-first-search, rounding down first.

Strong : Integer : 5 : strongBranching

Number of fractional variables which pseudocosts will be examinated before proceeding in the search tree.

Trust : Integer : 5 : TrustPseudoCosts

After computing pseudocosts many times for a variable trust previously computed pseudocosts and do not perform strong branching anymore (needs to be greater than Strong).

Expensive : Integer : 5 : ExpensiveStrong

Where to apply strong branching:

0 : normal;

In fractional variables :

1 : root node;

2 : depth less than modifier;

4 : if object == best possible;

6 : 2 and 4.

In all variables :

10 : root node;

>100 : when depth >= strategy/100 (otherwise 5).

4.4 Cut Generation

Cut application parameters. CBC cut generators can be configured to be applied only at root node or in the entire search tree. More flexible ways are also allowed. Possible values to indicate how often a cut generator should be applied are:

OFF : never try this cut;

ROOT : cuts applied only at root node;

IFMOVE : cuts will be used of they succeed in improving the dual bound;

FORCEON : forces the use of the cut generator at every node;

These parameters define cut application strategies and this parameter type will be denoted as **CUTAPPCHOICE**.

CUTS : LOGICAL : ON : CUTSONOFF

Cuts on (Cuts Off) activates (deactivates) all cuts.

GOMORY : CUTAPPCHOICE : IFMOVE : GOMORYCUTSCUTS

The original cutting planes proposed by Ralph Gomory.

LAGOMORY : CUTAPPCHOICE : OFF : LAGOMORYCUTS

Generates additional Gomory Cuts using Lagrangian Relaxation, as proposed in (Matteo Fischetti & Salvagnin, 2011). These cuts may be generated after all other cuts finished (END), which is the recommended option, and include the processing of "clean" cuts (like clique cuts) (ONLY) or any other integral valued cuts (CLEAN), resulting in the additional options: ENDONLYROOT, ENDCLEANROOT, ROOT, ENDONLY, ENDCLEAN, ENDBOTH, ONLYASWELL, CLEANASWELL, ONLYINSTEAD, BOTHASWELL, CLEANINSTEAD, BOTHINSTEAD, ONLYASWELLROOT, CLEANASWELLROOT e BOTHASWELLROOT.

LATWOMIR : CUTAPPCHOICE : OFF : LATWOMIRCUTS

Lagrangian MIR cuts from two rows. All options from lagomory except ROOT, ONLYASWELLROOT, CLEANASWELLROOT e BOTHASWELLROOT.

CLIQUE : CUTAPPCHOICE : IFMOVE : CLIQUECUTS

Determines the application of Clique cuts. These cuts are generated from conflicts(Atamtürk, Nemhauser, & Savelsbergh, 2000; Brito, Santos, & Poggi, 2015) between binary variables.

LIFT : CUTAPPCHOICE : OFF : LIFTANDPROJECTCUTS

Determines the application of Lift-and-Project cuts(Balas, Ceria, & Cornuéjols, 1993). These cuts can be relatively expensive to generate, so that they are off by default.

```
Mixed : CutAppChoice : IfMove : MixedIntegerRoundingCuts
```

Determines the application of MIR – Mixed Integer Rounding(Marchand & Wolsey, 2001) cuts.

```
two : CutAppChoice : Root : TwoMirCuts
```

Whether to use Two phase Mixed Integer Rounding(Dash & Günlük, 2004) cuts.

```
Knapsack : CutAppChoice : IfMove : KnapsackCuts
```

Determines the application of Knapsack cover cuts(Gu, Nemhauser, & Savelsbergh, 1998).

```
Flow : CutAppChoice : IfMove : FlowCoverCuts
```

Determines the application of lifted flow cover inequalities.

```
probing : CutAppChoice : forceOnStrong : ProbingCuts
```

Activates Probing Cuts. These cuts inspect the effect of fixing variables to different values. For this cut generator other more aggressive options are available:

forceOn onglobal forceonglobal forceOnBut forceOnStrong forceOnButStrong strongRoot

```
residual : CutAppChoice : Off : ResidualCapacityCuts
```

Switches residual capacity cuts.

```
Reduce : CutAppChoice : Off : ReduceAndSplit
```

Switches the Reduce-and-Split cuts as implemented by Francois Margot.

```
Reduce2 : CutAppChoice : Off : Reduce2AndSplit
```

Switches the Reduce-and-Split cuts as implemented by Giacomo Nannicini.

```
CutD : Integer : -1 : CutDepth
```

Activate cuts whenever the depth in the three is a multiple of CutD. When CutD=-1, cbc decides by itself if cuts should be applied or not.

```
CutL : Integer : -1 : CutLength
```

Gomory cuts can produce very dense rows which can slowdown the search. This option allows one to limit the maximum number of acceptable columns in gomory cuts. By default, cbc decides it (-1). Values greater than 0 indicate:

`0 <= CutL < 10,000,000` : maximum length of CutL for cuts generated at root node and in the tree;

`CutL >= 10,000,000` : allows cuts with unlimited length at root node, with a limit inside the tree. for example: `CutL=10,000,130` indicate that in the tree only cuts with at most 130 variables will be accepted.

passC : Integer : -1 : PassCuts

Maximum number of cut passes in the root note. If -1, the following strategy is used, according to the number of columns n.:

- $n \leq 500$: 100 passes;
- $500 < n \leq 5000$: 100 passes, stopping when bound improvements are small;
- $n > 5000$: 20 passes for larger problems.

Positive values for `PassC` indicate a maximum number of passes but cut generation will be interrupted if no bound improvement occurs. Entering a negative values v smaller than -1 for `PassC` forces the cut generation to continue to $|v|$ passes.

Zero : CutAppChoice : IfMove : ZeroHalfCuts

Activates the {0,1/2} cuts. These cuts are obtained after multiplying constraints by 0 or 0.5 and rounding.

4.5 Heuristics

Heuristic application.

Several heuristics accept a parameter which indicate where it will be activated. This parameter will be denoted by **HeurAppChoice**. Valid values for this parameter are:

`Off` : never apply;

`On` : applies heuristic after preprocessing;

`Before` : applies heuristic before preprocessing;

`Both` : applies in both cases;

heur : Logical : On : HeuristicsOnOff

Heuristics on (Heuristics off) activates (deactivates) all heuristics at once.

Round : HeurAppChoice : On : RoundingHeuristic

This switches on a simple (but effective) rounding heuristic at each node of tree.

Feas : HeurAppChoice : On : FeasibilityPump

This switches on feasibility pump heuristic at root. This is due to Fischetti, Lodi and Glover and uses a sequence of LPs to try and get an integer feasible solution.

PassF : Integer : 20 : PassFeasibilityPump

Indicates the maximum number os passes for the Feasibility Pump heuristic. Try higher values if no feasible solution was obtained.

Fraction : Double : 0.5 : FractionForaBAB

After a pass in feasibility pump, variables which have not moved about are fixed and if the preprocessed model is small enough a few nodes of branch and bound are done on reduced problem. Small problem has to be less than this fraction of the original.

Local : HeurAppChoice : Off : LocalTreeSearch

This switches on a local search algorithm when a solution is found. This is from Fischetti and Lodi and is not really a heuristic although it can be used as one.

```
PivotAndC : HeurAppChoice : Off : PivotAndComplement
```

Switches Pivot and Complement heuristic.

```
PivotAndF : HeurAppChoice : Off : PivotAndFix
```

Switches Pivot and Fix heuristic.

```
Combine : HeurAppChoice : On : CombineSolutions
```

This switches on a heuristic which does branch and cut on the problem given by just using variables which have appeared in one or more solutions. It obviously only tries after the discovery of two or more solutions.

```
Dins : HeurAppChoice : Off : CombineSolutions
```

Switches on the Distance Induced Neighborhood Search heuristic.

```
Combine2 : HeurAppChoice : Off : CombineSolutions
```

Same as before, but considers only variables which have the same value in two or more solutions.

```
Proximity : HeurAppChoice : Off : ProximitySearch
```

Controls the activation of the Proximity Search Heuristic.

```
Randomi : HeurAppChoice : Off : RandomizedRounding
```

Controls the activation of the Randomized Rounding Heuristic.

```
Rins : HeurAppChoice : On : Rins
```

Controls the activation of Relaxation Induced Neighborhood Search heuristic.

```
Rens : HeurAppChoice : Off : Rens
```

Controls the activation of Relaxation Enforced Neighborhood Search heuristic.

```
Vnd : HeurAppChoice : Off : VndVariableNeighborhoodSearch
```

Controls the activation of Variable Neighborhood Search heuristic.

```
DivingG : HeurAppChoice : Off : DivingGuided
```

Switches Guided Dives heuristic.

```
DivingP : HeurAppChoice : Off : DivingPseudoCost
```

Switches Diving heuristic using pseudocosts.

```
DivingF : HeurAppChoice : Off : DivingFractional
```

Switches Diving Fractional heuristic.

```
DivingS : HeurAppChoice : Off : DivingSome
```

Switches on a random diving heuristic at various times.

DiveO : Integer : -1 : DiveOpt

Controls the application of diving heuristic:

>2 and <20 :

- `3` : only at root node and if no solution;
- `4` : only at root and if this heuristic has not got solution;
- `5` : decay only if no solution;
- `6` : if depth <3 or decay;
- `7` : run up to 2 times if solution found 4 otherwise.

`>10 AND <20` : all only at root;

`>20` : all with value-20.

DiveS : Integer : 100 : DiveSolves

Diving solve options:

`>0` : performs up to this many solves, last digit is ignores and used for extra options:

- `1` : allow fixing of satisfied integers (but not at bound);
- `2` : switch off above permanently if goes infeasible.

4.6 Limits and Tolerances

IntegerT : Double : 1e-6 : IntegerTolerance

Solutions are only considered feasible if for all integer variables their current value is no more than this tolerance away from the closest integer.

Sec : Integer : Infinity : Seconds

Time limit for execution. See also `TimeM`.

TimeM : String : cpu : TimeMode

How the processing time is measured in cbc: `ELAPSED` considers the wallclock time. `Cpu` computes the total Cpu time used. When running in parallel the cpu time advances much faster than the elapsed time.

MaxN : Integer : Infinity : MaxNodes

Maximum number of nodes in the search tree. Can be used to avoid very large memory requirements.

MaxSo : Integer : Infinity : MaxSolutions

Maximum number of integer solutions. If you are interested in any feasible solution, not necessarily the optimal one, set it to 1.

CutO : Double : Infinity : CutOff

Specifies an upper bound (in a minimization sense) for the solution cost. All nodes in the branch-and-bound where the lower bound is worse than this value will be pruned.

Constraint : Logical : Off : ConstraintFromCutoff

When on appends to the model a constraint in the format: objective function ≤ CutOff (or best solution found). This can be useful since additional cuts and implications may be generated from this constraint.

Ratio : Double : 0 : RatioGap

If the gap between the best solution and the best possible solution is less than this fraction then the search is terminated. Use allowableGap for absolute values.

5 . References

Atamtürk, A., Nemhauser, G. L., & Savelsbergh, M. W. P. (2000). Conflict graphs in solving integer programming problems. *European Journal of Operational Research*, *121*, 40–55.

Balas, E., Ceria, S., & Cornuéjols, G. (1993). A lift-and-project cutting plane algorithm for mixed 0-1 programs. *Mathematical Programming*, *58*(1–3), 295–324. https://doi.org/10.1007/BF01581273

Brito, S. S., Santos, H. G., & Poggi, M. (2015). A Computational Study of Conflict Graphs and Aggressive Cut Separation in Integer Programming. *Electronic Notes in Discrete Mathematics*, *50*, 355–360. https://doi.org/10.1016/j.endm.2015.07.059

Dash, S., & Günlük, O. (2004). Valid Inequalities Based on Simple Mixed-Integer Sets. In *Integer Programming and Combinatorial Optimization* (Vol. 3064, pp. 33–45).

Fischetti, M., Glover, F., & Lodi, A. (2005). The feasibility pump. *Mathematical Programming*, *104*, 91–104.

Fischetti, M., & Salvagnin, D. (2011). A relax-and-cut framework for gomory mixed-integer cuts. *Mathematical Programming Computation*, *3*(2), 79–102. https://doi.org/10.1007/s12532-011-0024-x

Gomory, R. E. (1958). Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, *64*(5), 275–278.

Gu, Z., Nemhauser, G. L., & Savelsbergh, M. W. P. (1998). Lifted Cover Inequalities for 0-1 Integer Programs: Computation. *INFORMS Journal on Computing*, *10*(4), 427–437. https://doi.org/10.1287/ijoc.1100.0396

Kutschka, M. (2008). Algorithms to Separate { 0 , 1 2 } -Chvátal-Gomory Cuts. https://doi.org/10.1007/s00453-008-9218-7

Marchand, H., & Wolsey, L. A. (2001). Aggregation and Mixed Integer Rounding to Solve MIPs. *Operations Research*, *49*(3), 363–371. https://doi.org/http://dx.doi.org/10.1287/opre.2013.1200