BONMIN Users' Manual

Pierre Bonami and Jon Lee

September 22, 2009

1 Introduction

BONMIN (Basic Open-source Nonlinear Mixed INteger programming) is an open-source code for solving general MINLP (Mixed Integer NonLinear Programming) problems. It is distributed on COIN-OR (www.coin-or.org) under the CPL (Common Public License). The CPL is a license approved by the OSI[1], (Open Source Initiative), thus BONMIN is OSI Certified Open Source Software.

There are several algorithmic choices that can be selected with BONMIN. B-BB is a NLP-based branch-and-bound algorithm, B-OA is an outer-approximation decomposition algorithm, B-QG is an implementation of Quesada and Grossmann's branch-and-cut algorithm, and B-Hyb is a hybrid outer-approximation based branch-and-cut algorithm.

Some of the algorithmic choices require the ability to solve MILP (Mixed Integer Linear Programming) problems and NLP (NonLinear Programming) problems. The default solvers for these are, respectively, the COIN-OR codes Cbc and Ipopt. In turn, Cbc uses further COIN-OR modules: Clp (for LP (Linear Programming) problems), Cgl (for generating MILP cutting planes), as well as various other utilities. It is also possible to step outside the open-source realm and use Cplex as the MILP solver. We expect to make an interface to other NLP solvers as well.

Additional documentation is availble on the Bonmin wiki at

https://projects.coin-or.org/Bonmin

Types of problems solved

BONMIN solves MINLPs of the form

[1] http://www.opensource.org

$$
\begin{aligned}
&\min f(x)\\
&\text{s.t.}\\
&g^L \leq g(x) \leq g^U,\\
&x^L \leq x \leq x^U,\\
&x \in \mathbb{R}^n,\ x_i \in \mathbb{Z}\ \forall i \in I,
\end{aligned}
$$

where the functions $f : \{x \in \mathbb{R}^n : x^L \leq x \leq x^U\} \rightarrow \mathbb{R}$ and $g : \{x \in \mathbb{R}^n : x^L \leq x \leq x^U\} \rightarrow \mathbb{R}^m$ are assumed to be twice continuously differentiable, and $I \subseteq \{1, \ldots, n\}$. We emphasize that `BONMIN` treats problems that are cast in *minimization* form.

The different methods that `BONMIN` implements are exact algorithms when the functions f and g are convex but are only heuristics when this is not the case (i.e., `BONMIN` is not a *global* optimizer).

Algorithms

`BONMIN` implements four different algorithms for solving MINLPs:

- `B-BB`: a simple branch-and-bound algorithm based on solving a continuous nonlinear program at each node of the search tree and branching on variables [2]; we also allow the possibility of SOS (Type 1) branching
- `B-OA`: an outer-approximation based decomposition algorithm [3, 4]
- `B-QG`: an outer-approximation based branch-and-bound algorithm [7]
- `B-Hyb`: a hybrid outer-approximation/nonlinear programming based branch-and-cut algorithm [1]

In this manual, we will not go into a further description of these algorithms. Mathematical details of these algorithms and some details of their implementations can be found in [1] .

Whether or not you are interested in the details of the algorithms, you certainly want to know which one of these four algorithms you should choose to solve your particular problem. For convex MINLPs, experiments we have made on a reasonably large test set of problems point in favor of using `B-Hyb` (it solved the most of the problems in our test set in 3 hours of computing time). Therefore, it is the default algorithm in `BONMIN`. Nevertheless, there are cases where `B-OA` is much faster than `B-Hyb` and others where `B-BB` is interesting. `B-QG` corresponds mainly to a specific parameter setting of `B-Hyb` where some features are disabled. For nonconvex MINLPs, we strongly recommend using `B-BB` (the outer-approximation algorithms have not been tailored to treat nonconvex problems at this point). Although even `B-BB` is only a heuristic for such problems, we have added several options to try and improve the quality of the solutions it provides (see Section 5.3).

Required third party code

In order to run BONMIN, you have to download other external libraries (and pay attention to their licenses!):

- Lapack (Linear Algebra PACKage)
- Blas (Basic Linear Algebra Subroutines)
- a sparse linear solver that is supported by Ipopt, e.g., MA27 from the HSL (Harwell Subroutine Library), MUMPS, or Pardiso

Note that Lapack and the Blas are free for commercial use from the Netlib Repository[2], but they are not OSI Certified Open Source Software. The linear solver MA27 is freely available for noncommercial use.

The above software is sufficient to run BONMIN as a stand-alone C++ code, but it does not provide a modeling language. For functionality from a modeling language, BONMIN can be invoked from Ampl[3] (no extra installation is required provided that you have a licensed copy of Ampl installed), though you need the ASL (Ampl Solver Library) which is obtainable from the Netlib.

BONMIN can use FilterSQP [?] as an alternative to Ipopt for solving NLPs.

Also, in the outer approximation decomposition method B-OA, some MILP problems are solved. By default BONMIN uses Cbc to solve them, but it can also be set up to use the commercial solver Cplex[4].

Tested platforms

BONMIN has been installed on the following systems:

- Linux using g++ version 3.* and 4.* and Intel 9.* and 10.*
- Windows using version Cygwin 1.5.18
- Mac OS X using gcc 3.* and 4.* and Intel 9.* and 10.*
- SunOS 5 using gcc 4.3

2 Obtaining BONMIN

The BONMIN package consists of the source code for the BONMIN project but also source code from other COIN-OR projects:

- BuildTools
- Cbc
- Cgl

[2] http://www.netlib.org
[3] http://www.ampl.com
[4] http://www.ilog.com/products/cplex/product/mip.cfm

- Clp
- CoinUtils
- Ipopt
- Osi

When downloading the BONMIN package you will download the source code for all these and libraries of problems to test the codes.

Before downloading BONMIN you need to know which branch of Bonmin you want to download. In particular you need to know if you want to download the latest version from:

- the Stable branch, or from
- the Released branch.

These different version are made according to the guidelines of COIN-OR. The interpretation of these guidelines for the Bonmin project is explained on the wiki pages of Bonmin.

The main distinction between the Stable and Release branch is that a stable version that we propose to download may evolve over time to include bug fixes while a released version will never change. The released versions present an advantage in particular if you want to make experiments which you want to be able to reproduce the stable version presents the advantage that it is less work for you to update in the event where we fix a bug.

The easiest way to obtain the released version is by downloading a compressed archive from Bonmin archive directory. The latest release is Bonmin-1.1.0.

The only way to obain one of the stable versions is through subversion.

In Unix[5]-like environments, to download the latest stable version of Bonmin (1.1) in a sub-directory, say Bonmin-1.1 issue the following command

```
svn co https://projects.coin-or.org/svn/Bonmin/stable/1.1 Bonmin-1.1
```

This copies all the necessary COIN-OR files to compile BONMIN to Bonmin-1.1. To download BONMIN using svn on Windows, follow the instructions provided at COIN-OR.

2.1 Obtaining required third party code

BONMIN needs a few external packages which are not included in the BONMIN package.

- Lapack (Linear Algebra PACKage)

[5]UNIX is a registered trademark of The Open Group.

- Blas (Basic Linear Algebra Subroutines)
- A sparse linear solver.
- Optionally ASL (the Ampl Solver Library), to be able to use BONMIN from Ampl.

Since these third-party software modules are released under licenses that are incompatible with the CPL, they cannot be included for distribution with BONMIN from COIN-OR, but you will find scripts to help you download them in the subdirectory ThirdParty of the BONMIN distribution. In most Linux distribution and CYGWIN, Lapack and Blas are available as prebuilt binary packages in the distribution (and are probably already installed on your machine).

Linear solvers are used by Ipopt. The most up-to-date information regarding the supported linear solvers and how to install them is found in Section 2.2 of the Ipopt manual. Several options are available for linear solvers: MA27 from the Harwell Subroutine Library (and optionally, but strongly recommended, MC19 to enable automatic scaling in Ipopt), MA57 or Mumps. In our experiment MA27 and MA57 usualy perform significantly better but they are freely available only for non-commercial, academic use. Note that all linear solver can also use Metis.

3 Installing BONMIN

The build process for BONMIN should be fairly automatic as it uses GNU autotools. It has been successfully compiled and run on the following platforms:

- Linux using g++ version 3.4 and 4.0
- Windows using version Cygwin 1.5.18
- Mac OS X using gcc 3.4 and 4.0

For Cygwin and OS X some specific setup has to be done prior to instalation. These step are described on the wiki pages of Bonmin CygwinInstall[6] and OsxInstall[7].

BONMIN is compiled and installed using the commands:

```
./configure -C
make
make install
```

[6] https://projects.coin-or.org/Bonmin/wiki/CygwinInstall
[7] https://projects.coin-or.org/Bonmin/wiki/OsxInstall

This installs the executable `bonmin` in `coin-Bonmin/bin`. In what follows, we assume that you have put the executable `bonmin` on your path.

The `configure` script attempts to find all of the machine specific settings (compiler, libraries,...) necessary to compile and run the code. Although `configure` should find most of the standard ones, you may have to manually specify a few of the settings. The options for the configure script can be found by issuing the command

```
./configure --help
```

For a more in depth description of these options, the reader is invited to refer to the COIN-OR `BuildTools` trac page[8].

3.1 Specifying the location of `Cplex` libraries

If you have `Cplex` installed on your machine, you may want to use it as the Mixed Integer Linear Programming subsolver in `B-OA` and `B-Hyb`. To do so you have to specify the location of the header files and libraries. You can either specify the location of the header files directory by passing it as an argument to the configure script or by writing it into a `config.site` file.

In the former case, specify the location of the `Cplex` header files by using the argument `--with-cplexincdir` and the location of the `Cplex` library with `--with-cplexlib` (note that on the Linux platform you will also need to add `-lpthread` as an argument to `--with-cplexlib`).

For example, on a Linux machine if `Cplex` is installed in `/usr/ilog` , you would invoke configure with the arguments as follows:

```
./configure --with-cplex-incdir=/usr/ilog/cplex/include/ilcplex \
  --with-cplex-lib="/usr/ilog/cplex/lib/libcplex.a -lpthread"
```

In the latter case, put a file called `config.site` in a subdirectory named `share` of the installation directory (if you do not specify an alternate installation directory to the `configure` script with the `--prefix` argument, the installation directory is the directory where you execute the `configure` script). To specify the location of `Cplex` , insert the following lines in the `config.site` file:

```
with_cplex_lib="/usr/ilog/cplex/lib/libcplex.a -lpthread"
with_cplex_incdir="/usr/ilog/cplex/include/ilcplex"
```

[8] https://projects.coin-or.org/BuildTools

(You will find a `config.site` example in the subdirectory `BuildTools` of `coin-Bonmin`.)

3.2 Compiling BONMIN in a external directory

It is possible to compile `BONMIN` in a directory different from `coin-Bonmin`. This is convenient if you want to have several executables compiled for different a rchitectures or have several executables compiled with different options (debugging and production, shared and static libraries).

To do this just create a new directory, for example `Bonmin-build` in the parent directory of `coin-Bonmin` and run the configure command from `Bonmin-build`:

```
../Bonmin-0.99/configure -C
```

This will create the makefiles in `coin-Bonmin`, and you can then compile with the usual `make` and `make install` (in `Bonmin-build`).

3.3 Building the documentation

The documentation for `BONMIN` consists of a users' manual (this document) and a reference manual. You can build a local copy of the reference manual provided that you have Latex and Doxygen installed on your machine. Issue the command `make doxydoc` in `coin-Bonmin`. It calls Doxygen to build a copy of the reference manual. An html version of the reference manual can then be accessed in `doc/html/index.html`.

3.4 Running the test programs

By issuing the command `make test` , you build and run the automatic test program for `BONMIN`.

4 Running BONMIN

`BONMIN` can be run

(i) from a command line on a `.nl` file (see [6]),

(ii) from the modeling language `Ampl`[9] (see [8]),

(iii) from the Gams[10] modeling language,

(iv) by invoking it from a C/C++ program.

[9] http://www.ampl.com
[10] http://www.gams.com/

(v) remotely through the NEOS[11] web interface.

In the subsections that follow, we give some details about the various ways to run BONMIN.

4.1 On a .nl file

BONMIN can read a .nl file which could be generated by Ampl (for example mytoy.nl in the Bonmin-dist/Bonmin/test subdirectory). The command line takes just one argument which is the name of the .nl file to be processed.

For example, if you want to solve mytoy.nl, from the Bonmin-dist directory, issue the command:

```
bonmin test/mytoy.nl
```

4.2 From Ampl

To use BONMIN from Ampl you just need to have the directory where the bonmin executable is in your $PATH and to issue the command

```
option solver bonmin;
```

in the Ampl environment. Then the next solve will use BONMIN to solve the model loaded in Ampl. After the optimization is finished, the values of the variables in the best-known or optimal solution can be accessed in Ampl. If the optimization is interrupted with <CTRL-C> the best known solution is accessible (this feature is not available in Cygwin).

4.2.1 Example Ampl model

simple Ampl example model follows:

```
# An Ampl version of toy

reset;

var x binary;
var z integer >= 0 <= 5;
var y{1..2} >=0;
minimize cost:
```

[11] http://neos.mcs.anl.gov/neos

```
    - x - y[1] - y[2] ;

subject to
    c1: ( y[1] - 1/2 )^2 + (y[2] - 1/2)^2 <= 1/4 ;
    c2: x - y[1] <= 0 ;
    c3: x + y[2] + z <= 2;

option solver bonmin; # Choose BONMIN as the solver (assuming that
                      # bonmin is in your PATH

solve;                # Solve the model
display x;
display y;
```

(This example can be found in the subdirectory `Bonmin/examples/amplExamples/` of the `BONMIN` package.)

4.2.2 Setting up branching priorities, directions and declaring SOS1 constraints in ampl

Branching priorities, branching directions and pseudo-costs can be passed using `Ampl` suffixes. The suffix for branching priorities is `"priority"` (variables with a higher priority will be chosen first for branching), for branching direction is `"direction"` (if direction is 1 the $\geq$ branch is explored first, if direction is -1 the $\leq$ branch is explored first), for up and down pseudo costs `"upPseudoCost"` and `"downPseudoCost"` respectively (note that if only one of the up and down pseudo-costs is set in the `Ampl` model it will be used for both up and down).

For example, to give branching priorities of 10 to variables `y` and 1 to variable `x` and to set the branching directions to explore the upper branch first for all variables in the simple example given, we add before the call to solve:

```
suffix priority IN, integer, >=0, <= 9999;
y[1].priority := 10;
y[2].priority := 10;
x.priority := 1;

suffix direction IN, integer, >=-1, <=1;
y[1].direction := 1;
y[2].direction := 1;
x.direction := 1;
```

SOS Type-1 branching is also available in `BONMIN` from `Ampl`. We follow the conventional way of doing this with suffixes. Two type of suffixes should be declared:

```
suffix sosno IN, integer, >=1;  # Note that the solver assumes that these
                                #   values are positive for SOS Type 1
suffix ref IN;
```

Next, suppose that we wish to have variables

```
var X {i in 1..M, j in 1..N} binary;
```

and the "convexity" constraints:

```
subject to Convexity {i in 1..M}:
   sum {j in 1..N} X[i,j] = 1;
```

(note that we must explicitly include the convexity constraints in the `Ampl` model).

Then after reading in the data, we set the suffix values:

```
# The numbers 'val[i,j]' are chosen typically as
#     the values 'represented' by the discrete choices.
let {i in 1..M, j in 1..N} X[i,j].ref := val[i,j];

# These identify which SOS constraint each variable belongs to.
let {i in 1..M, j in 1..N} X[i,j].sosno := i;
```

4.3 From `Gams`

Thanks to the GAMSlinks[12] project, Bonmin is available in `Gams` from release 22.5 of the `GAMS`[13] modeling system. The system is available for download from GAMS[14]. Without buying a license it works as a demo with limited capabilities. Documentation for using `BONMIN` in `GAMS` is available at

`http://www.gams.com/solvers/coin.pdf`

4.4 From a C/C++ program

`BONMIN` can also be run from within a C/C++ program if the user codes the functions to compute first- and second-order derivatives. An example of such a program is available in the subdirectory `CppExample` of the `examples` directory. For further explanations, please refer to the reference manual.

[12] http://projects.coin-or.org/GAMSlinks
[13] http://www.gams.com/
[14] http://download.gams-software.com/

5 Options

5.1 Passing options to BONMIN

Options in BONMIN can be set in several different ways.

First, you can set options by putting them in a file called bonmin.opt in the directory where bonmin is executing. If you are familiar with the file ipopt.opt (formerly named PARAMS.DAT) in Ipopt, the syntax of the bonmin.opt is similar. For those not familiar with ipopt.opt, the syntax is simply to put the name of the option followed by its value, with no more than two options on a single line. Anything on a line after a # symbol is ignored (i.e., treated as a comment).

Note that BONMIN sets options for Ipopt. If you want to set options for Ipopt (when used inside BONMIN) you have to set them in the file bonmin.opt (the standard Ipopt option file ipopt.opt is not read by BONMIN.) For a list and a description of all the Ipopt options, the reader may refer to the documentation of Ipopt[15].

Since bonmin.opt contains both Ipopt and BONMIN options, for clarity all BONMIN options should be preceded with the prefix "bonmin." in bonmin.opt . Note that some options can also be passed to the MILP subsolver used by BONMIN in the outer approximation decomposition and the hybrid (see Subsection 5.2).

The most important option in BONMIN is the choice of the solution algorithm. This can be set by using the option named bonmin.algorithm which can be set to B-BB, B-OA, B-QG, or B-Hyb (it's default value is B-BB). Depending on the value of this option, certain other options may be available or not. Table 1 gives the list of options together with their types, default values and availability in each of the four algorithms. The column labeled 'type' indicates the type of the parameter ('F' stands for float, 'I' for integer, and 'S' for string). The column labeled default indicates the global default value. Then for each of the four algorithm B-BB, B-OA, B-QG, and B-Hyb, '+' indicates that the option is available for that particular algorithm while '−' indicates that it is not.

An example of a bonmin.opt file including all the options with their default values is located in the Test sub-directory.

A small example is as follows:

```
bonmin.bb_log_level 4
bonmin.algorithm B-BB
print_level 6
```

This sets the level of output of the branch-and-bound in BONMIN to 4, the algorithm to branch-and-bound and the output level for Ipopt to 6.

When BONMIN is run from within Ampl, another way to set an option is through the internal Ampl command options. For example

[15] http://www.coin-or.org/Ipopt/documentation/node54.html

```
options bonmin_options "bonmin.bb_log-level 4 \
                  bonmin.algorithm B-BB print_level 6";
```

has the same affect as the `bonmin.opt` example above. Note that any `BONMIN` option specified in the file `bonmin.opt` overrides any setting of that option from within Ampl.

A third way is to set options directly in the C/C++ code when running `BONMIN` from inside a C/C++ program as is explained in the reference manual.

A detailed description of all of the `BONMIN` options is given in Appendix A. In the following, we give some more details on options for the MILP subsolver and on the options specifically designed for nonconvex problems.

Table 1: List of options and compatibility with the different algorithms.

Option	type	default	B-BB	B-OA	B-QG	B-Hyb
Algorithm choice						
algorithm	S	BBB	+	+	+	+
Bonmin ecp based strong branching						
ecp_abs_tol_strong	F	1e06	+	+	+	+
ecp_max_rounds_strong	I	0	+	+	+	+
ecp_rel_tol_strong	F	0.1	+	+	+	+
lp_strong_warmstart_method	S	Basis	+	+	+	+
Branch-and-bound options						
allowable_fraction_gap	F	0	+	+	+	+
allowable_gap	F	0	+	+	+	+
cutoff	F	1e+100	+	+	+	+
cutoff_decr	F	1e05	+	+	+	+
integer_tolerance	F	1e06	+	+	+	+
iteration_limit	I	INT_MAX	+	+	+	+
nlp_failure_behavior	S	stop	+	+	+	+
node_comparison	S	dynamic	+	+	+	+
node_limit	I	INT_MAX	+	+	+	+
num_cut_passes	I	1	-	+	+	+
num_cut_passes_at_root	I	20	-	+	+	+
number_before_trust	I	8	+	+	+	+
number_strong_branch	I	20	+	+	+	+
solution_limit	I	INT_MAX	+	+	+	+
sos_constraints	S	enable	+	-	+	+
time_limit	F	1e+10	+	+	+	+
tree_search_strategy	S	topnode	+	+	+	+
variable_selection	S	strongbranching	+	+	+	+
Diving options						
max_backtracks_in_dive	I	5	+	+	+	+
max_dive_depth	I	INT_MAX	+	+	+	+
stop_diving_on_cutoff	S	no	+	+	+	+
MILP cutting planes in hybrid						
continued on next page						

Option	type	default	B-BB	B-OA	B-QG	B-Hyb
2mir_cuts	I	0	-	+	-	+
Gomory_cuts	I	5	-	+	-	+
clique_cuts	I	5	-	+	-	+
cover_cuts	I	5	-	+	-	+
flow_covers_cuts	I	5	-	+	-	+
lift_and_project_cuts	I	0	-	+	-	+
mir_cuts	I	5	-	+	-	+
probing_cuts	I	5	-	+	-	+
reduce_and_split_cuts	I	0	-	+	-	+
Nlp solution robustness						
max_consecutive_failures	I	10	+	+	+	+
max_random_point_radius	F	100000	+	-	-	-
num_iterations_suspect	I	1	+	+	+	+
num_retry_unsolved_random_point	I	0	+	+	+	+
random_point_perturbation_interval	F	1	+	-	-	-
random_point_type	S	Jon	+	-	-	-
Nlp solve options in B-Hyb						
nlp_solve_frequency	I	10	-	-	-	+
nlp_solve_max_depth	I	10	-	-	-	+
nlp_solves_per_depth	F	1e+30	-	-	-	+
Options for MILP subsolver in OA decomposition						
milp_log_level	I	0	-	+	-	+
milp_subsolver	S	Cbc_D	-	+	-	+
Options for OA decomposition						
oa_dec_time_limit	F	30	+	+	+	+
oa_log_frequency	F	100	+	+	+	+
oa_log_level	I	1	+	+	+	+
Options for ecp cuts generation						
ecp_abs_tol	F	1e06	-	-	-	+
ecp_max_rounds	I	5	-	-	-	+
ecp_propability_factor	F	1000	-	-	-	+
ecp_rel_tol	F	0	-	-	-	+
filmint_ecp_cuts	I	0	-	-	-	+
Options for non-convex problems						
continued on next page						

Option	type	default	B-BB	B-OA	B-QG	B-Hyb
max_consecutive_infeasible	I	0	+	+	+	+
num_resolve_at_infeasibles	I	0	+	-	-	-
num_resolve_at_node	I	0	+	-	-	-
num_resolve_at_root	I	0	+	-	-	-
Outer Approximation cuts generation						
add_only_violated_oa	S	no	-	+	+	+
cut_strengthening_type	S	none	-	+	+	+
disjunctive_cut_type	S	none	-	+	+	+
oa_cuts_log_level	I	0	-	+	+	+
oa_cuts_scope	S	global	-	+	+	+
tiny_element	F	1e08	-	+	+	+
very_tiny_element	F	1e17	-	+	+	+
Output ond log-levels ptions						
bb_log_interval	I	100	+	-	+	+
bb_log_level	I	1	+	-	+	+
lp_log_level	I	0	-	-	+	+
Strong branching setup						
candidate_sort_criterion	S	bestpscost	+	+	+	+
maxmin_crit_have_sol	F	0.1	+	+	+	+
maxmin_crit_no_sol	F	0.7	+	+	+	+
min_number_strong_branch	I	0	+	+	+	+
number_before_trust_list	I	0	+	+	+	+
number_look_ahead	I	0	+	+	+	+
number_strong_branch_root	I	INT_MAX	+	+	+	+
setup_pseudo_frac	F	0.5	+	+	+	+
trust_strong_branching_for_pseudo_cost	S	yes	+	+	+	+
nlp interface option						
file_solution	S	no	+	+	+	+
nlp_log_level	I	1	+	+	+	+
nlp_solver	S	Ipopt	+	+	+	+
warm_start	S	none	+	-	-	-

5.2 Passing options to the MILP subsolver

In the context of outer approximation decomposition, a standard MILP solver is used. Several option are available for configuring this MILP solver. `BONMIN` allows a choice of different MILP solvers through the option `bonmin.milp_subsolver`. Values for this option are: `Cbc_D` which uses `Cbc` with its default settings, `Cplex` which uses `Cplex` with its default settings, and `Cbc_Par` which uses a version of `Cbc` that can be parameterized by the user.

The options that can be set are the node-selection strategy, the number of strong-branching candidates, the number of branches before pseudo costs are to be trusted, and the frequency of the various cut generators (options marked with * in Table 1). To pass those options to the MILP subsolver, you have to replace the prefix "`bonmin.`" with "`milp_sub.`".

5.3 Getting good solutions to nonconvex problems

To solve a problem with non-convex constraints, one should only use the branch-and-bound algorithm `B-BB`.

A few options have been designed in `BONMIN` specifically to treat problems that do not have a convex continuous relaxation. In such problems, the solutions obtained from `Ipopt` are not necessarily globally optimal, but are only locally optimal. Also the outer-approximation constraints are not necessarily valid inequalities for the problem.

No specific heuristic method for treating nonconvex problems is implemented yet within the OA framework. But for the pure branch-and-bound `B-BB`, we implemented a few options having in mind that lower bounds provided by `Ipopt` should not be trusted, and with the goal of trying to get good solutions. Such options are at a very experimental stage.

First, in the context of nonconvex problems, `Ipopt` may find different local optima when started from different starting points. The two options `num_resolve_at_root` and `num_resolve_at_node` allow for solving the root node or each node of the tree, respectively, with a user-specified number of different randomly-chosen starting points, saving the best solution found. Note that the function to generate a random starting point is very naïve: it chooses a random point (uniformly) between the bounds provided for the variable. In particular if there are some functions that can not be evaluated at some points of the domain, it may pick such points, and so it is not robust in that respect.

Secondly, since the solution given by `Ipopt` does not truly give a lower bound, we allow for changing the fathoming rule to continue branching even if the solution value to the current node is worse than the best-known solution. This is achieved by setting `allowable_gap` and `allowable_fraction_gap` and `cutoff_decr` to negative values.

5.4 Notes on `Ipopt` options

`Ipopt` has a very large number of options, to get a complete description of them,

you should refer to the Ipopt manual. Here we only mention and explain some of the options that have been more important to us, so far, in developing and using BONMIN.

5.4.1 Default options changed by BONMIN

Ipopt has been tailored to be more efficient when used in the context of the solution of a MINLP problem. In particular, we have tried to improve Ipopt's warm-starting capabilities and its ability to prove quickly that a subproblem is infeasible. For ordinary NLP problems, Ipopt does not use these options by default, but BONMIN automatically changes these options from their default values.

Note that options set by the user in `bonmin.opt` will override these settings.

`mu_strategy` and `mu_oracle` are set, respectively, to `adaptive` and `probing` by default (these are newly implemented strategies in Ipopt for updating the barrier parameter [9] which we have found to be more efficient in the context of MINLP).

`gamma_phi` and `gamma_theta` are set to 10^{-8} and 10^{-4} respectively. This has the effect of reducing the size of the filter in the line search performed by Ipopt.

`required_infeasibility_reduction` is set to 0.1. This increases the required infeasibility reduction when Ipopt enters the restoration phase and should thus help detect infeasible problems faster.

`expect_infeasible_problem` is set to `yes` which enables some heuristics to detect infeasible problems faster.

`warm_start_init_point` is set to `yes` when a full primal/dual starting point is available (generally all the optimizations after the continuous relaxation has been solved).

`print_level` is set to 0 by default to turn off Ipopt output.

5.4.2 Some useful Ipopt options

bound_relax_factor is by default set to 10^{-8} in Ipopt. All of the bounds of the problem are relaxed by this factor. This may cause some trouble when constraint functions can only be evaluated within their bounds. In such cases, this option should be set to 0.

References

[1] P. Bonami, A. Wächter, L.T. Biegler, A.R. Conn, G. Cornuéjols, I.E. Grossmann, C.D. Laird, J. Lee, A. Lodi, F. Margot and N. Sawaya. An algorithmic framework for convex mixed integer nonlinear programs. *Discrete Optimization* 5:186–204, 2008.

[2] O.K. Gupta and V. Ravindran. Branch and bound experiments in convex nonlinear integer programming. *Management Science*, 31:1533–1546, 1985.

[3] M. Duran and I.E. Grossmann. An outer-approximation algorithm for a class of mixed-integer nonlinear programs. *Mathematical Programming*, 36:307–339, 1986.

[4] R. Fletcher and S. Leyffer. Solving mixed integer nonlinear programs by outer approximation. *Mathematical Programming*, 66:327–349, 1994.

[5] R. Fletcher and S. Leyffer. User manual for filterSQP. *University of Dundee Numerical Analysis Report NA-181*, 1998.

[6] D.M. Gay. Writing **.nl** files. Sandia National Laboratories, Technical Report No. 2005-7907P, 2005.

[7] I. Quesada and I.E. Grossmann. An LP/NLP based branched and bound algorithm for convex MINLP optimization problems. *Computers and Chemical Engineering*, 16:937–947, 1992.

[8] R. Fourer and D.M. Gay and B.W. Kernighan. AMPL: A Modeling Language for Mathematical Programming, Second Edition, Duxbury Press Brooks Cole Publishing Co., 2003.

[9] J. Nocedal, A. Wächter, and R. A. Waltz. Adaptive Barrier Strategies for Nonlinear Interior Methods. Research Report RC 23563, IBM T. J. Watson Research Center, Yorktown, USA (March 2005; revised January 2006)

[10] A. Wächter and L. T. Biegler. On the Implementation of a Primal-Dual Interior Point Filter Line Search Algorithm for Large-Scale Nonlinear Programming. Mathematical Programming 106(1), pp. 25-57, 2006

A List of `BONMIN` options

A.1 Algorithm choice

algorithm: Choice of the algorithm.
This will preset some of the options of bonmin depending on the algorithm choice. The default value for this string option is "B-BB".
Possible values:

- B-BB: simple branch-and-bound algorithm,

- B-OA: OA Decomposition algorithm,
- B-QG: Quesada and Grossmann branch-and-cut algorithm,
- B-Hyb: hybrid outer approximation based branch-and-cut,
- B-Ecp: ecp cuts based branch-and-cut a la FilMINT.

A.2 Bonmin ecp based strong branching

ecp_abs_tol_strong: Set the absolute termination tolerance for ECP rounds in strong branching.
The valid range for this real option is $0 \leq$ `ecp_abs_tol_strong` $<$ `+inf` and its default value is $1 \cdot 10^{-06}$.

ecp_max_rounds_strong: Set the maximal number of rounds of ECP cuts in strong branching.
The valid range for this integer option is $0 \leq$ `ecp_max_rounds_strong` $<$ `+inf` and its default value is 0.

ecp_rel_tol_strong: Set the relative termination tolerance for ECP rounds in strong branching.
The valid range for this real option is $0 \leq$ `ecp_rel_tol_strong` $<$ `+inf` and its default value is 0.1.

lp_strong_warmstart_method: Choose method to use for warm starting lp in strong branching
(Advanced stuff) The default value for this string option is "Basis".
Possible values:

- Basis: Use optimal basis of node
- Clone: Clone optimal problem of node

A.3 Branch-and-bound options

allowable_fraction_gap: Specify the value of relative gap under which the algorithm stops.
Stop the tree search when the gap between the objective value of the best known solution and the best bound on the objective of any solution is less than this fraction of the absolute value of the best known solution value. The valid range for this real option is $-1 \cdot 10^{+20} \leq$ `allowable_fraction_gap` $\leq 1 \cdot 10^{+20}$ and its default value is 0.

allowable_gap: Specify the value of absolute gap under which the algorithm stops.
Stop the tree search when the gap between the objective value of the best known solution and the best bound on the objective of any solution is less than this. The valid range for this real option is $-1 \cdot 10^{+20} \leq$ `allowable_gap` $\leq 1 \cdot 10^{+20}$ and its default value is 0.

cutoff: Specify cutoff value.
cutoff should be the value of a feasible solution known by the user (if any). The algorithm will only look for solutions better than cutoof. The valid range for this real option is $-1 \cdot 10^{+100} \leq$ `cutoff` $\leq 1 \cdot 10^{+100}$ and its default value is $1 \cdot 10^{+100}$.

cutoff_decr: Specify cutoff decrement.
Specify the amount by which cutoff is decremented below a new best upper-bound (usually a small positive value but in non-convex problems it may be a negative value). The valid range for this real option is $-1 \cdot 10^{+10} \leq$ `cutoff_decr` $\leq 1 \cdot 10^{+10}$ and its default value is $1 \cdot 10^{-05}$.

integer_tolerance: Set integer tolerance.
Any number within that value of an integer is considered integer. The valid range for this real option is $0 <$ `integer_tolerance` $<$ `+inf` and its default value is $1 \cdot 10^{-06}$.

iteration_limit: Set the cumulated maximum number of iteration in the algorithm used to process nodes continuous relaxations in the branch-and-bound.
value 0 deactivates option. The valid range for this integer option is $0 \leq$ `iteration_limit` $<$ `+inf` and its default value is 2147483647.

nlp_failure_behavior: Set the behavior when an NLP or a series of NLP are unsolved by Ipopt (we call unsolved an NLP for which Ipopt is not able to guarantee optimality within the specified tolerances).
If set to "fathom", the algorithm will fathom the node when Ipopt fails to find a solution to the nlp at that node whithin the specified tolerances. The algorithm then becomes a heuristic, and the user will be warned that the solution might not be optimal. The default value for this string option is "stop".
Possible values:

- stop: Stop when failure happens.
- fathom: Continue when failure happens.

node_comparison: Choose the node selection strategy.
Choose the strategy for selecting the next node to be processed. The default value for this string option is "best-bound".
Possible values:

- best-bound: choose node with the smallest bound,
- depth-first: Perform depth first search,
- breadth-first: Perform breadth first search,
- dynamic: Cbc dynamic strategy (starts with a depth first search and turn to best bound after 3 integer feasible solutions have been found).
- best-guess: choose node with smallest guessed integer solution

node_limit: Set the maximum number of nodes explored in the branch-and-bound search.
The valid range for this integer option is $0 \leq$ `node_limit` $<$ `+inf` and its default value is 2147483647.

num_cut_passes: Set the maximum number of cut passes at regular nodes of the branch-and-cut.
The valid range for this integer option is $0 \leq$ `num_cut_passes` $<$ `+inf` and its default value is 1.

num_cut_passes_at_root: Set the maximum number of cut passes at regular nodes of the branch-and-cut.
The valid range for this integer option is $0 \leq$ `num_cut_passes_at_root` $<$ `+inf` and its default value is 20.

number_before_trust: Set the number of branches on a variable before its pseudo costs are to be believed in dynamic strong branching.
A value of 0 disables pseudo costs. The valid range for this integer option is $0 \leq$ `number_before_trust` $<$ `+inf` and its default value is 8.

number_strong_branch: Choose the maximum number of variables considered for strong branching.
Set the number of variables on which to do strong branching. The valid range for this integer option is $0 \leq$ `number_strong_branch` $<$ `+inf` and its default value is 20.

solution_limit: Abort after that much integer feasible solution have been found by algorithm
value 0 deactivates option The valid range for this integer option is $0 \leq$ `solution_limit` $<$ `+inf` and its default value is 2147483647.

sos_constraints: Wether or not to activate SOS constraints.
(only type 1 SOS are supported at the moment) The default value for this string option is "enable".
Possible values:

- enable:
- disable:

time_limit: Set the global maximum computation time (in secs) for the algorithm.
The valid range for this real option is $0 <$ `time_limit` $< +$`inf` and its default value is $1 \cdot 10^{+10}$.

tree_search_strategy: Pick a strategy for traversing the tree
All strategies can be used in conjunction with any of the node comparison functions.Options which affect dfs-dive are max-backtracks-in-dive and max-dive-depth. The dfs-dive won't work in a non-convex problem where objective does not decrease down branches. The default value for this string option is "probed-dive".
Possible values:

- top-node: Always pick the top node as sorted by the node comparison function
- dive: Dive in the tree if possible, otherwise pick top node as sorted by the tree comparison function.
- probed-dive: Dive in the tree exploring two childs before continuing the dive at each level.
- dfs-dive: Dive in the tree if possible doing a depth first search.Backtrack on leaves or when a prescribed depth is attained or when estimate of best possible integer feasible solution in subtree is worst than cutoff. Once a prescribed limit of backtracks is attained pick top node as sorted by the tree comparison function
- dfs-dive-dynamic: Same as dfs-dive but once enough solution are found switch to best-bound and if too many nodes switch to depth-first.

variable_selection: Chooses variable selection strategy

The default value for this string option is "strong-branching".
Possible values:

- most-fractional: Choose most fractional variable
- strong-branching: Perform strong branching

- reliability-branching: Use reliability branching
- curvature-estimator: Use curvature estimation to select branching variable
- qp-strong-branching: Perform strong branching with QP approximation
- lp-strong-branching: Perform strong branching with LP approximation
- nlp-strong-branching: Perform strong branching with NLP approximation
- osi-simple: Osi method to do simple branching
- osi-strong: Osi method to do strong branching
- random: Method to choose branching variable randomly

A.4 Diving options

max_backtracks_in_dive: Set the number of backtracks in a dive when using dfs-dive tree searchstrategy.
The valid range for this integer option is $0 \leq$ `max_backtracks_in_dive` $<$ `+inf` and its default value is 5.

max_dive_depth: When using dfs-dive search. Maximum depth to go to from the diving board (node where the diving started.
The valid range for this integer option is $0 \leq$ `max_dive_depth` $<$ `+inf` and its default value is 2147483647.

stop_diving_on_cutoff: Flag indicating whether we stop diving based on guessed feasible objective and the current cutoff

The default value for this string option is "no".
Possible values:

- no:
- yes:

A.5 MILP cutting planes in hybrid

2mir_cuts: Frequency (in terms of nodes) for generating 2-MIR cuts in branch-and-cut
If k ¿ 0, cuts are generated every k nodes, if -99 ¡ k ¡ 0 cuts are generated every -k nodes but Cbc may decide to stop generating cuts, if not enough are generated at the root node, if k=-99 generate cuts only at the root node, if k=0 or 100 do not generate cuts. The valid range for this integer option is $-100 \leq$ `2mir_cuts` $<$ `+inf` and its default value is 0.

Gomory_cuts: Frequency k (in terms of nodes) for generating Gomory cuts in branch-and-cut.
If k ¿ 0, cuts are generated every k nodes, if -99 ¡ k ¡ 0 cuts are generated every -k nodes but Cbc may decide to stop generating cuts, if not enough are generated at the root node, if k=-99 generate cuts only at the root node, if k=0 or 100 do not generate cuts. The valid range for this integer option is $-100 \leq$ `Gomory_cuts` $<$ `+inf` and its default value is -5.

clique_cuts: Frequency (in terms of nodes) for generating clique cuts in branch-and-cut
If k ¿ 0, cuts are generated every k nodes, if -99 ¡ k ¡ 0 cuts are generated every -k nodes but Cbc may decide to stop generating cuts, if not enough are generated at the root node, if k=-99 generate cuts only at the root node, if k=0 or 100 do not generate cuts. The valid range for this integer option is $-100 \leq$ `clique_cuts` $<$ `+inf` and its default value is -5.

cover_cuts: Frequency (in terms of nodes) for generating cover cuts in branch-and-cut
If k ¿ 0, cuts are generated every k nodes, if -99 ¡ k ¡ 0 cuts are generated every -k nodes but Cbc may decide to stop generating cuts, if not enough are generated at the root node, if k=-99 generate cuts only at the root node, if k=0 or 100 do not generate cuts. The valid range for this integer option is $-100 \leq$ `cover_cuts` $<$ `+inf` and its default value is -5.

flow_covers_cuts: Frequency (in terms of nodes) for generating flow cover cuts in branch-and-cut
If k ¿ 0, cuts are generated every k nodes, if -99 ¡ k ¡ 0 cuts are generated every -k nodes but Cbc may decide to stop generating cuts, if not enough are generated at the root node, if k=-99 generate cuts only at the root node, if k=0 or 100 do not generate cuts. The valid range for this integer option is $-100 \leq$ `flow_covers_cuts` $<$ `+inf` and its default value is -5.

lift_and_project_cuts: Frequency (in terms of nodes) for generating lift-and-project cuts in branch-and-cut
If k ¿ 0, cuts are generated every k nodes, if -99 ¡ k ¡ 0 cuts are generated every -k nodes but Cbc may decide to stop generating cuts, if not enough are generated at the root node, if k=-99 generate cuts only at the root node, if k=0 or 100 do not generate cuts. The valid range for this integer option is $-100 \leq$ `lift_and_project_cuts` $<$ `+inf` and its default value is 0.

mir_cuts: Frequency (in terms of nodes) for generating MIR cuts in branch-and-cut
If k ¿ 0, cuts are generated every k nodes, if -99 ¡ k ¡ 0 cuts are generated every -k nodes but Cbc may decide to stop generating cuts, if not enough are generated at the root node, if k=-99 generate cuts only at the root node, if

k=0 or 100 do not generate cuts. The valid range for this integer option is $-100 \leq$ `mir_cuts` $<$ `+inf` and its default value is -5.

reduce_and_split_cuts: Frequency (in terms of nodes) for generating reduce-and-split cuts in branch-and-cut
If k ¿ 0, cuts are generated every k nodes, if -99 ¡ k ¡ 0 cuts are generated every -k nodes but Cbc may decide to stop generating cuts, if not enough are generated at the root node, if k=-99 generate cuts only at the root node, if k=0 or 100 do not generate cuts. The valid range for this integer option is $-100 \leq$ `reduce_and_split_cuts` $<$ `+inf` and its default value is 0.

A.6 Nlp solution robustness

max_consecutive_failures: (temporarily removed) Number n of consecutive unsolved problems before aborting a branch of the tree.
When $n > 0$, continue exploring a branch of the tree until n consecutive problems in the branch are unsolved (we call unsolved a problem for which Ipopt can not guarantee optimality within the specified tolerances). The valid range for this integer option is $0 \leq$ `max_consecutive_failures` $<$ `+inf` and its default value is 10.

max_random_point_radius: Set max value r for coordinate of a random point.
When picking a random point coordinate i will be in the interval [min(max(l,-r),u-r), max(min(u,r),l+r)] (where l is the lower bound for the variable and u is its upper bound) The valid range for this real option is $0 <$ `max_random_point_radius` $<$ `+inf` and its default value is 100000.

num_iterations_suspect: Number of iterations over which a node is considered "suspect" (for debugging purposes only, see detailed documentation).
When the number of iterations to solve a node is above this number, the subproblem at this node is considered to be suspect and it will be outputed in a file (set to -1 to deactivate this). The valid range for this integer option is $-1 \leq$ `num_iterations_suspect` $<$ `+inf` and its default value is -1.

num_retry_unsolved_random_point: Number k of times that the algorithm will try to resolve an unsolved NLP with a random starting point (we call unsolved an NLP for which Ipopt is not able to guarantee optimality within the specified tolerances).
When Ipopt fails to solve a continuous NLP sub-problem, if $k > 0$, the algorithm will try again to solve the failed NLP with k new randomly chosen starting points or until the problem is solved with success. The valid range for this integer option is $0 \leq$ `num_retry_unsolved_random_point` $<$ `+inf` and its default value is 0.

random_point_perturbation_interval: Amount by which starting point is perturbed when choosing to pick random point by perturbating starting point
The valid range for this real option is $0 <$ `random_point_perturbation_interval` $<$ `+inf` and its default value is 1.

random_point_type: method to choose a random starting point

The default value for this string option is "Jon".
Possible values:

- Jon: Choose random point uniformly between the bounds
- Andreas: perturb the starting point of the problem within a prescribed interval
- Claudia: perturb the starting point using the perturbation radius suffix information

A.7 Nlp solve options in B-Hyb

nlp_solve_frequency: Specify the frequency (in terms of nodes) at which NLP relaxations are solved in B-Hyb.
A frequency of 0 amounts to to never solve the NLP relaxation. The valid range for this integer option is $0 \leq$ `nlp_solve_frequency` $<$ `+inf` and its default value is 10.

nlp_solve_max_depth: Set maximum depth in the tree at which NLP relaxations are solved in B-Hyb.
A depth of 0 amounts to to never solve the NLP relaxation. The valid range for this integer option is $0 \leq$ `nlp_solve_max_depth` $<$ `+inf` and its default value is 10.

nlp_solves_per_depth: Set average number of nodes in the tree at which NLP relaxations are solved in B-Hyb for each depth.
The valid range for this real option is $0 \leq$ `nlp_solves_per_depth` $<$ `+inf` and its default value is $1 \cdot 10^{+30}$.

A.8 Options for MILP subsolver in OA decomposition

milp_log_level: specify MILP subsolver log level.
Set the level of output of the MILP subsolver in OA : 0 - none, 1 - minimal, 2 - normal low, 3 - normal high The valid range for this integer option is $0 \leq$ `milp_log_level` ≤ 3 and its default value is 0.

milp_subsolver: Choose the subsolver to solve MILP sub-problems in OA decompositions.
To use Cplex, a valid license is required and you should have compiled OsiCpx in COIN-OR (see Osi documentation). The default value for this string option is "Cbc_D".
Possible values:

- Cbc_D: Coin Branch and Cut with its default
- Cbc_Par: Coin Branch and Cut with passed parameters
- Cplex: Ilog Cplex

A.9 Options for OA decomposition

oa_dec_time_limit: Specify the maximum number of seconds spent overall in OA decomposition iterations.
The valid range for this real option is $0 \leq$ `oa_dec_time_limit` $<$ `+inf` and its default value is 30.

oa_log_frequency: display an update on lower and upper bounds in OA every n seconds
The valid range for this real option is $0 <$ `oa_log_frequency` $<$ `+inf` and its default value is 100.

oa_log_level: specify OA iterations log level.
Set the level of output of OA decomposition solver : 0 - none, 1 - normal, 2 - verbose The valid range for this integer option is $0 \leq$ `oa_log_level` ≤ 2 and its default value is 1.

A.10 Options for ecp cuts generation

ecp_abs_tol: Set the absolute termination tolerance for ECP rounds.
The valid range for this real option is $0 \leq$ `ecp_abs_tol` $<$ `+inf` and its default value is $1 \cdot 10^{-06}$.

ecp_max_rounds: Set the maximal number of rounds of ECP cuts.
The valid range for this integer option is $0 \leq$ `ecp_max_rounds` $<$ `+inf` and its default value is 5.

ecp_propability_factor: Factor appearing in formula for skipping ECP cuts.

Choosing -1 disables the skipping. The valid range for this real option is $-$`inf` $<$ `ecp_propability_factor` $<$ `+inf` and its default value is 1000.

ecp_rel_tol: Set the relative termination tolerance for ECP rounds.
The valid range for this real option is $0 \leq$ `ecp_rel_tol` $<$ `+inf` and its default value is 0.

filmint_ecp_cuts: Specify the frequency (in terms of nodes) at which some a la filmint ecp cuts are generated.
A frequency of 0 amounts to to never solve the NLP relaxation. The valid range for this integer option is $0 \leq$ `filmint_ecp_cuts` $<$ `+inf` and its default value is 0.

A.11 Options for non-convex problems

max_consecutive_infeasible: Number of consecutive infeasible subproblems before aborting a branch.
Will continue exploring a branch of the tree until "max_consecutive_infeasible"consecutive problems are infeasibles by the NLP sub-solver. The valid range for this integer option is $0 \leq$ `max_consecutive_infeasible` $<$ `+inf` and its default value is 0.

num_resolve_at_infeasibles: Number k of tries to resolve an infeasible node (other than the root) of the tree with different starting point.
The algorithm will solve all the infeasible nodes with k different random starting points and will keep the best local optimum found. The valid range for this integer option is $0 \leq$ `num_resolve_at_infeasibles` $<$ `+inf` and its default value is 0.

num_resolve_at_node: Number k of tries to resolve a node (other than the root) of the tree with different starting point.
The algorithm will solve all the nodes with k different random starting points and will keep the best local optimum found. The valid range for this integer option is $0 \leq$ `num_resolve_at_node` $<$ `+inf` and its default value is 0.

num_resolve_at_root: Number k of tries to resolve the root node with different starting points.
The algorithm will solve the root node with k random starting points and will keep the best local optimum found. The valid range for this integer option is $0 \leq$ `num_resolve_at_root` $<$ `+inf` and its default value is 0.

A.12 Outer Approximation cuts generation

add_only_violated_oa: Do we add all OA cuts or only the ones violated by current point?

The default value for this string option is "no".
Possible values:

- no: Add all cuts

- yes: Add only violated Cuts

cut_strengthening_type: Determines if and what kind of cut strengthening should be performed.

The default value for this string option is "none".
Possible values:

- none: No strengthening of cuts.
- sglobal: Strengthen global cuts.
- uglobal-slocal: Unstrengthened global and strengthened local cuts
- sglobal-slocal: Strengthened global and strengthened local cuts

disjunctive_cut_type: Determine if and what kind of disjunctive cuts should be computed.

The default value for this string option is "none".
Possible values:

- none: No disjunctive cuts.
- most-fractional: If discrete variables present, compute disjunction for most-fractional variable

oa_cuts_log_level: level of log when generating OA cuts.
0: outputs nothing, 1: when a cut is generated, its violation and index of row from which it originates, 2: always output violation of the cut. 3: output generated cuts incidence vectors. The valid range for this integer option is $0 \leq$ `oa_cuts_log_level` $<$ `+inf` and its default value is 0.

oa_cuts_scope: Specify if OA cuts added are to be set globally or locally valid

The default value for this string option is "global".
Possible values:

- local: Cuts are treated as globally valid
- global: Cuts are treated as locally valid

tiny_element: Value for tiny element in OA cut
We will remove "cleanly" (by relaxing cut) an element lower than this. The valid range for this real option is $-0 \leq$ `tiny_element` $<$ `+inf` and its default value is $1 \cdot 10^{-08}$.

very_tiny_element: Value for very tiny element in OA cut
Algorithm will take the risk of neglecting an element lower than this. The valid range for this real option is $-0 \leq$ `very_tiny_element` $<$ `+inf` and its default value is $1 \cdot 10^{-17}$.

A.13 Output ond log-levels options

bb_log_interval: Interval at which node level output is printed.
Set the interval (in terms of number of nodes) at which a log on node resolutions (consisting of lower and upper bounds) is given. The valid range for this integer option is $0 \leq$ `bb_log_interval` $<$ `+inf` and its default value is 100.

bb_log_level: specify main branch-and-bound log level.
Set the level of output of the branch-and-bound : 0 - none, 1 - minimal, 2 - normal low, 3 - normal high The valid range for this integer option is $0 \leq$ `bb_log_level` ≤ 5 and its default value is 1.

lp_log_level: specify LP log level.
Set the level of output of the linear programming sub-solver in B-Hyb or B-QG : 0 - none, 1 - minimal, 2 - normal low, 3 - normal high, 4 - verbose The valid range for this integer option is $0 \leq$ `lp_log_level` ≤ 4 and its default value is 0.

A.14 Strong branching setup

candidate_sort_criterion: Choice of the criterion to choose candidates in strong-branching

The default value for this string option is "best-ps-cost".
Possible values:

- best-ps-cost: Sort by decreasing pseudo-cost
- worst-ps-cost: Sort by increasing pseudo-cost
- most-fractional: Sort by decreasing integer infeasibility
- least-fractional: Sort by increasing integer infeasibility

maxmin_crit_have_sol: Weight towards minimum in of lower and upper branching estimates when a solution has been found.
The valid range for this real option is $0 \leq$ `maxmin_crit_have_sol` ≤ 1 and its default value is 0.1.

maxmin_crit_no_sol: Weight towards minimum in of lower and upper branching estimates when no solution has been found yet.
The valid range for this real option is $0 \leq$ `maxmin_crit_no_sol` ≤ 1 and its default value is 0.7.

min_number_strong_branch: Sets minimum number of variables for strong branching (overriding trust)
The valid range for this integer option is $0 \leq$ `min_number_strong_branch` $<$ `+inf` and its default value is 0.

number_before_trust_list: Set the number of branches on a variable before its pseudo costs are to be believed during setup of strong branching candidate list.
The default value is that of "number_before_trust" The valid range for this integer option is $-1 \leq$ `number_before_trust_list` $<$ `+inf` and its default value is 0.

number_look_ahead: Sets limit of look-ahead strong-branching trials
The valid range for this integer option is $0 \leq$ `number_look_ahead` $<$ `+inf` and its default value is 0.

number_strong_branch_root: Maximum number of variables considered for strong branching in root node.
The valid range for this integer option is $0 \leq$ `number_strong_branch_root` $<$ `+inf` and its default value is 2147483647.

setup_pseudo_frac: Proportion of strong branching list that has to be taken from most-integer-infeasible list.
The valid range for this real option is $0 \leq$ `setup_pseudo_frac` ≤ 1 and its default value is 0.5.

trust_strong_branching_for_pseudo_cost: Wether or not to trust strong branching results for updating pseudo costs.

The default value for this string option is "yes".
Possible values:

- no:
- yes:

A.15 nlp interface option

file_solution: Write a file bonmin.sol with the solution

The default value for this string option is "no".
Possible values:

- yes:
- no:

nlp_log_level: specify NLP solver interface log level (independent from ipopt print_level).
Set the level of output of the OsiTMINLPInterface : 0 - none, 1 - normal, 2 - verbose The valid range for this integer option is $0 \leq$ `nlp_log_level` ≤ 2 and its default value is 1.

nlp_solver: Choice of the solver for local optima of continuous nlp's

The default value for this string option is "Ipopt".
Possible values:

- Ipopt: Interior Point OPTimizer (https://projects.coin-or.org/Ipopt)
- filterSQP: Sequential quadratic programming trust region algorithm (http://www-unix.mcs.anl.gov/ leyffer/solvers.h- tml)
- all: run all available solvers at each node

warm_start: Select the warm start method
This will affect the function getWarmStart(), and as a consequence the warm starting in the various algorithms. The default value for this string option is "none".
Possible values:

- none: No warm start
- optimum: Warm start with direct parent optimum
- interior_point: Warm start with an interior point of direct parent