BONMIN Users' Manual

Pierre Bonami and Jon Lee

August 24, 2007

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

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

where the functions $f : \{x \in \mathbb{R}^n : x^L \le x \le x^U\} \rightarrow \mathbb{R}$ and $g : \{x \in \mathbb{R}^n : x^L \le x \le x^U\} \rightarrow \mathbb{R}^m$ 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 [6]
- `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)
- the sparse linear solver MA27 from the HSL (Harwell Subroutine Library)

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.

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.*
- Windows using version Cygwin 1.5.18
- Mac OS X using gcc 3.* and 4.*

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
- Clp
- CoinUtils

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

- 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. The latest release is Bonmin-0.1.0.

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 (0.1) in a sub-directory, say `Bonmin-0.1` issue the following command

```
svn co https://projects.coin-or.org/svn/Bonmin/stable/0.1 Bonmin-0.1
```

This copies all the necessary COIN-OR files to compile BONMIN to `Bonmin-0.1`. 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)
- Blas (Basic Linear Algebra Subroutines)
- the sparse linear solver MA27 from the Harwell Subroutine Library and optionally (but strongly recommended) MC19 to enable automatic scaling in Ipopt.

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

- 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[6]. For details on how to obtain these package, refer to the instructions in Section 2.2 of the Ipopt manual.

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[7] and OsxInstall[8].

BONMIN is compiled and installed using the commands:

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

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[9].

[6] 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).

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

[8] https://projects.coin-or.org/Bonmin/wiki/OsxInstall

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

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"
```

(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-0.1/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 [5]),

(ii) from the modeling language `Ampl`[10] (see [7]),

(iii) from the Gams[11] modeling language,

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

(v) remotely through the NEOS[12] 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
```

[10] http://www.ampl.com
[11] http://www.gams.com/
[12] http://neos.mcs.anl.gov/neos

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:
    - 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[13] project, Bonmin is available in Gams from release 22.5 of the GAMS[14] modeling system. The system is available for download from GAMS[15]. 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.

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[16].

Since bonmin.opt contains both Ipopt and BONMIN options, for clarity all BONMIN options should be preceded with the prefix "bonmin." in bonmin.opt .

[13] http://projects.coin-or.org/GAMSlinks
[14] http://www.gams.com/
[15] http://download.gams-software.com/
[16] http://www.coin-or.org/Ipopt/documentation/node54.html

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-Hyb). 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

```
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 Âmpl.

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 B. 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
output options						
bb_log_level	I	1	+	−	+	+
bb_log_interval	I	100	+	−	+	+
lp_log_level	I	0	−	−	+	+
milp_log_level	I	0	−	+	−	+
oa_log_level	I	1	−	+	−	+
oa_log_frequency	I	100	−	+	−	+
nlp_log_level	I	1	+	+	+	+
print_user_options	S	no	+	+	+	+
branch-and-bound options						
time_limit	F	10^{10}	+	+	+	+
allowable_gap	F	0	+	+	+	+
allowable_fraction_gap	F	0	+	+	+	+
cutoff	F	10^{100}	+	+	+	+
cutoff_decr	F	10^{-5}	+	+	+	+
integer_tolerance	F	10^{-6}	+	+	+	+
node_limit	I	INT_MAX	+	+	+	+
nodeselect_stra*	S	best-bound	+	+	+	+
number_before_trust*	I	8	−	+	+	+
number_strong_branch*	I	20	−	+	+	+
sos_constraints	S	enable	+	-	-	-
options for robustness						
max_random_point_radius	F	10^5	+	+	+	+
max_consecutive_failures	I	10	+	−	−	−
nlp_failure_behavior	S	stop	+	+	+	+
num_iterations_suspect	I	−1	+	+	+	+
num_retry_unsolved_random_point	I	0	+	+	+	+
options for nonconvex problems						
max_consecutive_infeasible	I	0	+	−	−	−
num_resolve_at_node	I	0	+	+	+	+
num_resolve_at_root	I	0	+	+	+	+
`B-Hyb` specific options						
nlp_solve_frequency	I	10	−	−	−	+
oa_dec_time_limit	F	120	−	−	−	+
tiny_element	F	10^{-8}	−	+	+	+
very_tiny_element	F	10^{-17}	−	+	+	+
MILP options						
cover_cuts*	I	−5	−	+	−	+
Gomory_cuts*	I	−5	−	+	−	+
milp_subsolver	S	`Cbc_D`	−	+	−	+
mir_cuts*	I	−5	−	+	−	+
probing_cuts[1]*	I	-5	−	+	−	+

* option is available for MILP subsolver (it is only passed if the `milp_subsolver` option is set to `Cbc_Par`, see Subsection 5.2).

[1] disabled for stability reasons.

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

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 [8] 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. IBM Research Report RC23771, Oct. 2005.

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

[6] 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.

[7] 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.

[8] 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)

[9] 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

B List of `BONMIN` options

B.1 `BONMIN` output options

bb_log_level specify branch-and-bound's 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

$$0 \leq \texttt{bb\_log\_level} \leq 3$$

and its default value is `1`.

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

$$0 \leq \texttt{bb\_log\_interval} < \infty$$

and its default value is `100`.

lp_log_level specify LP log level.
Set the level of output of the linear programming subsolver 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

$$0 \leq \texttt{lp\_log\_level} \leq 4$$

and its default value is `0`.

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

$$0 \leq \texttt{milp\_log\_level} \leq 3$$

and its default value is `0`.

oa_log_level specify OA iterations log level.
Set the level of output of OA decomposition solver :

- `0` - none,
- `1` - normal low,
- `2` - normal high.

The valid range for this integer option is

$$0 \leq \texttt{oa\_log\_level} \leq 2$$

and its default value is `1`.

oa_log_frequency specify OA log frequency. The valid range for this real option is

$$0 \leq \texttt{oa\_log\_frequency} \leq \infty$$

and its default value is 100.

nlp_log_level specify NLP solver interface log level (independent from ipopt print_level).
Set the level of output of the IpoptInterface :

- `0` - none,
- `1` - low and readable with warnings,
- `2` - verbose

The valid range for this integer option is

$$0 \leq \texttt{nlp\_log\_level} \leq 2$$

and its default value is `1`.

print_user_options Prints the list of options set by the user. The default value for this option is "no".
Possible values are:

- yes: print the list,
- no: don't.

B.2 BONMIN branch-and-bound options

algorithm Choice of the algorithm.

This will preset default values for most options of BONMIN but depending on which algorithm some of these can be changed (refer to Table 1 to see which options are valid with which algorithm). The default value for this string option is "B-Hyb".
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.

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 lower bound on the objective of any solution is less than this. The valid range for this real option is

$$-10^{20} \leq \texttt{allowable\_gap} \leq 10^{20}$$

and its default value is 0.

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

$$-10^{20} \leq \texttt{allowable\_fraction\_gap} \leq 10^{20}$$

and its default value is 0.

cutoff Specify a 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 (meaning with a lower objective value) than cutoff. The valid range for this real option is

$$-10^{100} \leq \texttt{cutoff} \leq 10^{100}$$

and its default value is 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

$$-10^{10} \leq \texttt{cutoff\_decr} \leq 10^{10}$$

and its default value is 10^{-05}.

nodeselect_stra 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 least bound,
- `depth-first`: Perform depth-first search,
- `breadth-first`: Perform breadth-first search,
- `dynamic`: `Cbc` dynamic strategy (start with depth-first search and turn to best bound after 3 integer feasible solutions have been found).

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

$$0 \leq \texttt{number\_strong\_branch} < \infty$$

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 dynamic strong branching. The valid range for this integer option is

$$0 \leq \texttt{number\_before\_trust} < \infty$$

and its default value is `8`.

time_limit Set the global maximum computation time (in seconds) for the algorithm.

The valid range for this real option is

$$0 < \texttt{time\_limit} < \infty$$

and its default value is 10^{+10}.

node_limit Set the maximum number of nodes explored in the branch-and-bound search.

The valid range for this integer option is

$$0 \leq \texttt{node\_limit} < \infty$$

and its default value is `INT_MAX` (as defined in system `limits.h`).

integer_tolerance Set integer tolerance.

Any number within that value of an integer is considered integer. The valid range for this real option is

$$0 < \texttt{integer\_tolerance} < 0.5$$

and its default value is 10^{-6}.

warm_start Select the warm start method. Possible values:

- `none`: no warm start,
- `optimum`: warm start with direct parent optimum",
- `interior_point`: Warm start with an interior point of direct parent".

The default value is `optimum`.

sos_constraints Wether or not to activate SOS constraints branching. Possible values are

- `enable`,
- `disable`.

The default value is `enable`.

B.3 BONMIN options for robustness

max_random_point_radius Set max value r for coordinate of a random point.

When picking a random point, each coordinate is selected uniformly 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. Beware that this is a very naive procedure. In particular, it may not be possible to evaluate some functions (such as log, 1/x) at such a randomly generated point (if BONMIN finds that this is the case, it will give up random point generation). The valid range for this real option is

$$0 < \texttt{max\_random\_point\_radius} < \infty$$

and its default value is 10^5.

max_consecutive_failures 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 (i.e., for which `Ipopt` can not guarantee optimality within the specified tolerances). The valid range for this integer option is

$$0 \le \texttt{max\_consecutive\_failures} < \infty$$

and its default value is `10`.

num_iterations_suspect (for debugging purposes only) number of iterations to consider a problem suspect.

When the number of iterations taken by the continuous nonlinear solver (for the moment this is Ipopt) to solve a node is above this number, the subproblem is considered to be suspect and is outputed to a file. If set to `-1` no subproblem is ever considered suspect. The valid range for this integer option is

$$-1 \le \texttt{num\_iterations\_suspect} < \infty$$

and its default value is -1.

nlp_failure_behavior Set the behavior when an NLP or a series of NLP are unsolved by `Ipopt` (an NLP is unsolved if `Ipopt` is not able to guarantee optimality within the specified tolerances).

If set to "`fathom`", the algorithm will fathom the node when an NLP is unsolved. The algorithm then becomes a heuristic. A warning that the solution might not be optimal is printed. The default value for this string option is "`stop`".

Possible values:

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

num_retry_unsolved_random_point Number k of times that the algorithm tries to resolve an unsolved NLP with a random starting point (unsolved NLP as defined above). When an NLP is unsolved, if $k > 0$, the algorithm tries 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

$$0 \le \texttt{num\_retry\_unsolved\_random\_point} < \infty$$

and its default value is `0`.

B.4 BONMIN options for non-convex problems

max_consecutive_infeasible Number k of consecutive infeasible subproblems before aborting a branch.

Explores a branch of the tree until k consecutive problems are infeasible by the NLP subsolver. The valid range for this integer option is

$$0 \le \texttt{max\_consecutive\_infeasible} < \infty$$

and its default value is 0.

num_resolve_at_root Number k of trials to solve the root node with different starting points.

The algorithm solves the root node with k random starting points and keeps the best local optimum found. The valid range for this integer option is

$$0 \le \texttt{num\_resolve\_at\_root} < \infty$$

and its default value is 0.

num_resolve_at_node Number k of tries to solve a node (other than the root) of the tree with different starting point.

The algorithm solves all the nodes with k different random starting points and keeps the best local optimum found. The valid range for this integer option is

$$0 \le \texttt{num\_resolve\_at\_node} < \infty$$

and its default value is 0.

B.5 BONMIN options : B-Hyb specific options

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 never solve the NLP relaxation. The valid range for this integer option is

$$0 \le \texttt{nlp\_solve\_frequency} < \infty$$

and its default value is 10.

oa_dec_time_limit Specify the maximum number of seconds spent overall in OA decomposition iterations.

The valid range for this real option is

$$0 \le \texttt{oa\_dec\_time\_limit} < \infty$$

and its default value is 120.

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

$$0 \leq \texttt{tiny\_element} < \infty$$

and its default value is 10^{-8}.

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

$$0 \leq \texttt{very\_tiny\_element} < \infty$$

and its default value is 10^{-17}.

milp_subsolver Choose the subsolver to solve MILPs 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-OR Branch and Cut with default options,
- `Cbc_Par`: COIN-OR Branch and Cut with options passed by user,
- `Cplex`: Ilog Cplex.

B.5.1 Cut generators frequency

For each one of the cut generators

Gomory_cuts

probing_cuts (by default probing cuts are currently disabled for numerical stability reason)

cover_cuts

mir_cuts Sets the frequency (in terms of nodes) for generating cuts of the given type in the branch-and-cut.

- $k > 0$, cuts are generated every k nodes,
- $-99 < k < 0$, cuts are generated every -k nodes but `Cbc` may decide to stop generating cuts, if not enough are generated at the root node,

- $k = -99$ cuts are generated only at the root node,
- $k = 0$ or $k = -100$ cuts are not generated.

The valid range for this integer option is

$$-100 \leq k < \infty$$

and its default value is -5.