BONMIN Users' Manual

Pierre Bonami and Jon Lee

June 20, 2008

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 [**?**]; we also allow the possibility of SOS (Type 1) branching
- `B-OA`: an outer-approximation based decomposition algorithm [**?**, **?**]
- `B-QG`: an outer-approximation based branch-and-bound algorithm [**?**]
- `B-Hyb`: a hybrid outer-approximation/nonlinear programming based branch-and-cut algorithm [**?**]

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 [**?**] .

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 **??**).

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 from Bonmin archive directory. The latest release is Bonmin-0.99.0.

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 (0.99) in a sub-directory, say Bonmin-0.99 issue the following command

```
svn co https://projects.coin-or.org/svn/Bonmin/stable/0.99 Bonmin-0.99
```

This copies all the necessary COIN-OR files to compile BONMIN to Bonmin-0.99. 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)

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

- the sparse linear solver MA27 from the Harwell Subroutine Library and optionally (but strongly recommended) MC19 to enable automatic scaling in Ipopt.
- 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

[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

```
./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].

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`.)

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

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 [?]),

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

(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.

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

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:
    - 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

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

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 . 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 **??**).

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 **??** 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 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.

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

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