BARON user manual v. 2024.5.8

May 8, 2024

Nick Sahinidis, The Optimization Firm, LLC, niksah@minlp.com, http://www.minlp.com

Contents

1 Introduction **2**
1.1 Licensing and software requirements 2
2 Model requirements **3**
2.1 Allowable nonlinear functions 3
2.2 Variable and expression bounds 3
3 Installation **4**
3.1 Installing and running BARON under its own parser interface 4
3.2 Installing and running BARON under Pyomo or JuMP 4
3.3 Installing and running BARON under MATLAB 5
4 BARON input **5**
4.1 Usage 6
4.2 Input grammar 6
4.3 The options section 7
4.4 The problem data 7
4.5 Error messages 9
4.6 Sample input file 10
4.7 Other ways to access BARON 11
5 BARON output **11**
5.1 BARON screen output 11
5.2 Termination messages, model and solver statuses 13
5.3 BARON solution output 14
6 Some BARON features **16**
6.1 No starting point is required 17
6.2 Finding a few of the best or all feasible solutions 17
6.3 Using BARON as a multi-start heuristic solver 19
6.4 Systematic treatment of unbounded problems 20

6.5 Systematic treatment of infeasible problems 20
6.6 Handling of complementarity constraints 21
6.7 Parallel capabilities 21
7 The BARON options 21
7.1 Termination options 22
7.2 Relaxation options 24
7.3 Range reduction options 25
7.4 Tree management options 25
7.5 Local search options 26
7.6 Output and file name options 26
7.7 Subsolver options 27
7.8 Licensing options 29
7.9 Other options 29
8 Bibliography 31

1 Introduction

The Branch-And-Reduce Optimization Navigator (BARON) is a computational system for the *global solution* of algebraic nonlinear programs (NLPs) and mixed-integer nonlinear programs (MINLPs).

While traditional NLP and MINLP algorithms are only guaranteed to provide global optima under certain convexity assumptions, BARON implements deterministic global optimization algorithms of the branch-and-bound type that are *guaranteed to provide global optima* under fairly general assumptions. These assumptions include the existence of finite lower and upper bounds on nonlinear expressions in the NLP or MINLP to be solved.

BARON implements algorithms of the branch-and-bound type, enhanced with a variety of constraint propagation and duality techniques for reducing ranges of variables in the course of the algorithm.

Parts of the BARON software were created at the University of Illinois at Urbana-Champaign and Carnegie Mellon University.

1.1 Licensing and software requirements

The demo version of BARON is freely available from The Optimization Firm and can be downloaded from http://www.minlp.com/download. This code can handle problems with up to 10 variables, 10 constraints, and 50 nonlinear operations. In order to use BARON for larger problems, users will need to have a valid BARON license. The Optimization Firm provides licenses that permit users to use BARON directly on any Windows or Linux platform as well as under

JuMP, MATLAB, Pyomo, and YALMIP. In addition, BARON distributors AIMMS, AMPL and GAMS provide licenses for using BARON under their modeling systems.

The software includes the solvers CLP/CBC, FilterSD, FilterSQP, and IPOPT for solving BARON's linear/integer programming (LP/MIP) and nonlinear programming (NLP) subproblems. BARON also includes high-quality numerical software from HSL, a collection of Fortran codes for large-scale scientific computation–see http://www.hsl.rl.ac.uk/. In addition, BARON can utilize IBM's ILOG CPLEX for solving LP/MIP subproblems if CPLEX is installed on the user's computer as a dynamic library. Additional licensed LP/MIP and NLP solvers are available under the AIMMS, AMPL, and GAMS versions of BARON and may expedite convergence. Specifically, XPRESS can be used for LP/MIP and any NLP solver available under AIMMS, AMPL and GAMS can be used for NLLP. The list of NLP solvers currently includes CONOPT, MINOS, SNOPT and KNITRO, among others.

2 Model requirements

BARON addresses the problem of finding global solutions to general nonlinear and mixed-integer nonlinear programs:

$$\begin{array}{ll} \min & f(x) \\ \text{s.t.} & g(x) \leq 0 \\ & x \in X \end{array}$$

where $f : X \to \mathbb{R}$, $g : X \to \mathbb{R}^m$, and $X \subset \mathbb{R}^n$. The set X may include integer restrictions and the constraints may include complementarity relationships. All functions must be algebraic and provided explicitly. The types of functions f and g currently handled by BARON are discussed below.

2.1 Allowable nonlinear functions

In addition to multiplication and division, BARON can handle nonlinear functions that involve $\exp(x)$, $\ln(x)$, x^α for real α, and β^x for real β. AIMMS/BARON, AMPL/BARON, and GAMS/BARON automatically handle $|x|$ and x^y, where x and y are variables; otherwise, suitable transformations discussed below can be used. There is currently no support for other functions, including the trigonometric functions $\sin(x)$, $\cos(x)$, etc.

2.2 Variable and expression bounds

Nonlinear expressions in the mathematical program to be solved must be bounded below and/or above. It is important that finite lower and upper bounds be provided by the user for as many problem variables as possible. However, providing finite bounds for variables alone may not be enough to guarantee finite bounds on nonlinear expressions arising in the model. For example, consider the term $1/x$ for $x \in [0, 1]$, which has finite variable bounds, but is unbounded. It is important to provide bounds for problem variables that guarantee that the problem functions

are finitely-valued in the domain of interest. If the user model does not include variable bounds that guarantee that all nonlinear expressions are finitely-valued, BARON will attempt to infer appropriate bounds from problem constraints. If this step fails, global optimality of the solutions provided cannot be guaranteed.

3 Installation

If you intend to use BARON under GAMS, AIMMS, AMPL, or YALMIP, then BARON software already comes installed with your modeling system. In this case, simply follow your system's manual to find out how to use BARON. If you intend to access the stand-alone version of BARON, entirely on its own or via JuMP, MATLAB, or Pyomo, place the BARON executable and license in your PATH. There is no internet connection requirement or any other additional requirement for the license to function. Detailed instructions for these installations follow.

3.1 Installing and running BARON under its own parser interface

1. Go to http://minlp.com/baron-downloads and download BARON for your platform. On Windows, the baron-win64.exe file you download from our site is an installer that you need to run and will help you install the BARON executable. On other platforms, unzip the zip archive to obtain the BARON executable for your platform.
2. Put the BARON license file, the BARON executable and any dynamic libraries distributed with it somewhere in your system PATH
3. Consult this manual on how to call BARON from the command prompt.

For a silent (non-interactive) installation on Windows, download the installer and run it as follows from the command line:

```
baron-win64.exe /SILENT
```

If an installation log file is additionally desired, it can be generated by the command:

```
baron-win64.exe /SILENT /LOG=filename
```

where `filename` is the desired name for the log file. In both cases, Windows will still request permission to run the executable before the silent installer is launched.

3.2 Installing and running BARON under Pyomo or JuMP

1. Go to http://minlp.com/baron-downloads and download BARON for your platform. On Windows, the baron-win64.exe file you download from our site is an installer that you need to run and will help you install the BARON executable. On other platforms, unzip the zip archive to obtain the BARON executable for your platform.

2. Put the BARON license file, the BARON executable and any dynamic libraries distributed with it somewhere in your system PATH.

3. Consult the Pyomo or JuMP manuals on how to call BARON. A good place to start for Pyomo is at http://www.pyomo.org/workshop-examples/. For JuMP, see https://jump.readthedocs.io/en/latest/quickstart.html.

3.3 Installing and running BARON under MATLAB

1. Download the MATLAB/BARON interface from http://minlp.com/downloads/matbar/matbar.zip

2. Unzip and place the contents of matbar.zip in a location of your choice; this will create a directory named matbar

3. Go to http://minlp.com/baron-downloads and download BARON for your platform. On Windows, the baron-win64.exe file you download from our site is an installer that you need to run and will help you install the BARON executable. On other platforms, unzip the zip archive to obtain the BARON executable for your platform.

4. In the matbar directory, place your BARON executable and name it barin.exe on Windows platforms and barin on all other platforms. If any dynamic libraries are distributed with the BARON executable, make sure to place them in the same location with the BARON executable.

5. Add the matbar directory to your system PATH. Simply placing it in your MATLAB path may not suffice

6. Put the BARON license file somewhere in your system PATH, e.g., in your matbar directory

7. Run BARON_install from MATLAB

8. Consult the Help directory that comes with the MATLAB/BARON interface. It contains the interface and solver manuals.

4 BARON input

There are various ways to input an optimization problem to BARON:

- Directly, using the BARON modeling language.
- Indirectly, using one of the available BARON interfaces under AIMMS, AMPL, GAMS, JuMP, MATLAB, Pyomo, or YALMIP.

In this section, we describe the BARON modeling language in detail.

4.1 Usage

BARON provides a high-level modeling language capable of reading a mixed-integer nonlinear optimization model in a relatively simple format. Input is provided in the form of a text file. Even though it is not required, it is strongly recommended that all BARON input files have the extension `.bar`. Let the input file be called `test.bar` and let the name of the BARON executable be `baron`. Then, issuing the command

```
baron test
```

or

```
baron test.bar
```

results in BARON parsing the file and solving the problem.

4.2 Input grammar

The following rules should be followed when preparing a BARON input file:

- All statements should be terminated by a semicolon (`;`).
- Reserved words must appear in uppercase letters.
- Variable and equation names can be in lower or upper case. The parser is case sensitive, i.e., `X1` and `x1` are two different variables.
- Variable and equation names should be no longer than 128 characters.
- Variable and equation names must start with a letter.
- String options should be no longer than 250 characters.
- With the exception of underscores (_), non-alphanumeric characters such as hyphens (-) are not permitted in variable names.
- Any text between `//` and the end of a line is ignored (i.e., it is treated as a comment).
- The signs "+", "-", "*" and "/" have their usual meaning of arithmetic operations.
- "^" is the power/exponentiation operator where, if the base is a negative constant, the exponent must be an integer. Note that the order of operations may vary in different computing environments as illustrated in the following example.

  $x \hat{} y \hat{} z = (x \hat{} y) \hat{} z$ in GAMS, MATLAB, and Excel.

  $x \hat{} y \hat{} z = x \hat{} (y \hat{} z)$ in Fortran, AMPL, BARON, Mathematica.

- A unary operator following an arithmetic operator is rejected by various compilers and accepted by others. BARON permits such expressions and assumes parentheses following the arithmetic operator through the end of the expression. For instance:

$$x\hat{} - y * z * -w * v = x\hat{}(-y * z * (-w * v))$$

- The exponential function is denoted as exp().
- The natural logarithm is available as log() as well as ln(). To enter $\log_{10}()$ in the model, use the transformation $\log_{10}(x) = \log_{10}(\mathrm{e}) * \log(x) = 0.4342944819032518 * \log(x)$.
- BARON does not allow x^y, where x and y are both variables. It is permissible to have either x or y as a variable in this case but not both. The following reformulation can be used around this: $x^y = \exp(y * \log(x))$. This reformulation is done automatically when BARON is used under AIMMS, AMPL, GAMS, or MATLAB.
- BARON does not allow the use of absolute values $|x|$ in the model file. However, this function can be modeled equivalently as $|x| = (x^2)^{0.5}$. This reformulation is done automatically when BARON is used under AIMMS, AMPL, GAMS, or MATLAB.
- Parentheses ("(" and ")") can be used in any meaningful combination with operations in mathematical expressions.
- Constraints must contain at least one expression that does not evaluate to a constant.

The input file is divided into two sections: the options and the problem data sections.

4.3 The options section

This section is optional. If used, it should be placed before any other programmatic statements. Any of BARON's algorithmic options can be specified here. This section has the following form:

```
OPTIONS {
<optname1>: <optvalue1>;
<optname2>: <optvalue2>;
<optname3>: <optvalue3>;
}
```

The names and corresponding values of the BARON options are described in detail in Section 7. Options not specified here take their default values. Instead of `OPTIONS`, the word `OPTION` can also be used.

4.4 The problem data

This section contains the data relating to the particular problem to be solved. The section can be divided into the following parts. Note that the words `EQUATIONS`, `ROWS`, and `CONSTRAINTS` are used interchangeably.

- **Variable declaration**: All variables used in the problem have to be declared before they are used in equations. Variables can be declared as binary, integer, positive, or free using the keywords `BINARY_VARIABLES`, `INTEGER_VARIABLES`, `POSITIVE_VARIABLES`, and `VARIABLES` respectively. In these keywords, `VARIABLE` or `VAR` may be used instead of `VARIABLES` and the underscore may be replaced by a space. *All* discrete (binary and integer) variables should be declared before *any* continuous variables. A sample declaration is as follows:

```
BINARY_VARIABLES y1, y2;   // 0-1 variables
INTEGER_VARIABLES x3, x7;  // discrete variables
POSITIVE_VARIABLES x1, x4, x6; // nonnegative variables
VARIABLE x5;        // this is a free variable
```

- **Variable bounds** (optional): Lower and upper bounds on previously declared variables can be declared using the keywords `LOWER_BOUNDS` and `UPPER_BOUNDS`, respectively. The word `BOUND` can be used instead of `BOUNDS`. A sample bounds declaration follows:

```
LOWER_BOUNDS{
x7: 10;
x5: -300;
}

UPPER_BOUND{
x4: 100;
}
```

- **Branching priorities** (optional): Branching priorities can be provided using the keyword `BRANCHING_PRIORITIES`. The default values of these parameters are set to 1. Variable violations are multiplied by the user-provided priorities before a branching variable is selected. A sample branching priorities section follows:

```
BRANCHING_PRIORITIES{
x3: 10;
x5: 0; }
```

  The effect of this input is that variable `x3` will be given higher priority than all others, while variable `x5` will never be branched upon.

- **Equation declaration**: An identifier (name) corresponding to each equation (constraint) has to be declared first. The keywords `EQUATION` and `EQUATIONS` can be used for this purpose. A sample equation declaration is shown below.

```
EQUATIONS e1, e2, e3;
```

  The naming rules for equations are the same as those for variables, i.e., all equation names are case-sensitive and should begin with a letter.

- `RELAXATION_ONLY_EQUATIONS <list equation names>;`

  This equation declaration can be used to specify constraints to be used for relaxation construction only. This is optional and must follow after the `EQUATIONS` declaration and before the equation definitions.

- `CONVEX_EQUATIONS <list equation names>;`

  This equation declaration can be used to specify constraints that are convex. This is optional and must follow after the `EQUATIONS` declaration and before the equation definitions.

- **Equation definition**: Each equation (or inequality) declared above is written in this section of the input file. The equation is preceded by its corresponding identifier. The bounds on the equations can be specified using the symbols `==` (equal to), `<=` (less than or equal to) and `>=` (greater than or equal to). Both `<=` and `>=` can be used in the same equation. A sample equation definition is shown below.

  ```
  e1: 5*x3 + y2 - 3*x5^3 >= 1;
  e2: y1 + 2*x4 - 2*x7 == 25.7;
  e3: -20 <= x4 + 2*y1*x3 + x6 <= 50;
  ```

  Variables must appear only on one side of the relational operator. That is, the "left-hand side" and the "right-hand side" should be pure numbers or expressions involving constants but no variables.

- **Objective function**: BARON optimizes a given objective function. This can be declared using the `OBJ` and the `minimize` or `maximize` keywords. A sample objective definition is shown below:

  ```
  OBJ: minimize 7*x3 + 2*x6;
  ```

- **Starting point** (optional): A starting point can be optionally specified using the keyword `STARTING_POINT` as follows:

  ```
  STARTING_POINT{
  x1: 50;
  x4: 100;
  x7: 300;
  }
  ```

4.5 Error messages

Any errors in the input file are reported in the form of "warnings" and "errors." BARON tries to continue execution despite warnings. In the event that the warnings and/or errors are severe, the program execution is stopped and the line where the fatal error occurred is displayed. The input file should be checked even if the warnings are not severe as the problem might have been parsed in a way other than it was intended to be.

4.6 Sample input file

A sample input file for BARON is shown below:

```
//  This is a gear train design problem taken from the GAMS test library
//
//  A compound gear train is to be designed to achieve a specific
//  gear ratio between the driver and driven shafts. The objective
//  of the gear train design is to find the number of teeth of the
//  four gears and to obtain a required gear ratio of 1/6.931.
//
//  The problem originated from:
//  Deb, K, and Goyal, M, Optimizing Engineering Designs Using a
//  Combined Genetic Search. In Back, T, Ed, Proceedings of the
//  Seventh International Conference on Genetic Algorithms. 1997,
//  pp. 521-528.

INTEGER_VARIABLES  i1,i2,i3,i4;         // number of teeth in each of the gears

LOWER_BOUNDS{
i1: 12;
i2: 12;
i3: 12;
i4: 12;
}

UPPER_BOUNDS{
i1: 60;
i2: 60;
i3: 60;
i4: 60;
}

EQUATIONS  e2,e3;                       // symmetry constraints

e2:  - i3 + i4 >= 0;
e3:    i1 - i2 >= 0;

// the objective aimms to make the reciprocal of the
// gear ratio  as close to 6.931 as possible.
// an ideal design will have an objective equal to 1.

OBJ: minimize (6.931 - i1*i2/(i3*i4))^2 + 1;

STARTING_POINT{
i1: 24;
i2: 24;
```

```
i3: 24;
i4: 24;
}
```

Additional examples can be found at http://www.minlp.com/download.

4.7 Other ways to access BARON

For information on how to access BARON under MATLAB, see the BARON/MATLAB interface manual at http://www.minlp.com/downloads/matbar/matbar.zip. For information on how to access BARON under AIMMS, AMPL, GAMS, JuMP, Pyomo, or YALMIP, consult the corresponding websites of these modeling systems.

5 BARON output

5.1 BARON screen output

The screen output below is obtained for the MINLP model `gear.bar`.

```
===========================================================================
 BARON version 24.5.8. Built: WIN-64 Wed May 8 10:05:37 EDT 2024
 Running on machine PONTIOS

 BARON is a product of The Optimization Firm.
 For information on BARON, see https://minlp.com/about-baron
 License file baronlice.txt is not valid for this version of BARON.
 Continuing in demo mode.
 Model size is allowable within BARON demo size.

 If you publish work using this software, please cite publications from
 https://minlp.com/baron-publications, such as:

 Khajavirad, A. and N. V. Sahinidis,
 A hybrid LP/NLP paradigm for global optimization relaxations,
 Mathematical Programming Computation, 10, 383-421, 2018.
===========================================================================
 This BARON run may utilize the following subsolver(s)
 For LP/MIP/QP: CLP/CBC, ILOG CPLEX
 For NLP: IPOPT, FILTERSQP
===========================================================================
 Starting solution is feasible with a value of   36.1768
 Doing local search
 Solving bounding LP
 Starting multi-start local search
```

```
 Preprocessing found feasible solution with value 20.3805
 Preprocessing found feasible solution with value 1.04488
 Done with local search
===============================================================================
  Iteration    Open nodes         Time (s)    Lower bound      Upper bound
*         1             1             0.03      1.00000          1.00117
          1             1             0.03      1.00000          1.00117
*         5             3             0.03      1.00000          1.00006
*         8             4             0.03      1.00000          1.00002
*        11             3             0.03      1.00000          1.00001
*        32             0             0.03      1.00000          1.00000
         32             0             0.03      1.00000          1.00000

                         *** Normal completion ***

 Wall clock time:                    0.13
 Total CPU time used:                0.03

 Total no. of BaR iterations:      32
 Best solution found at node:      32
 Max. no. of nodes in memory:       5

 All done
===============================================================================
```

The solver first tests feasibility of the user-supplied starting point. If this point is found to be feasible, BARON prints a message to that effect along with the corresponding objective function value; no related message is printed if this point is infeasible. BARON subsequently performs a randomized local search procedure. Whenever an improved solution point is found during this process, BARON prints a message to that effect along with the corresponding objective function value. Execution then proceeds with branch-and-bound. Information is printed every 1,000,000 branch-and-bound iterations and every 30 seconds. Additionally, information is printed whenever the value of the incumbent is improved by at least 10^{-5} and at the end of the search. During branch-and-bound search, a star (*) in the first position of a line indicates that a better feasible solution was found that improves the value of previous best known solution by at least 10^{-5}. The log fields include the iteration number, number of open branch-and-bound nodes, the time taken thus far, the lower bound, and the upper bound for the problem. The log output fields are summarized below:

Field	Description
`Iteration`	The number of the current iteration. A plus (`+`) following the iteration number denotes reporting while solving a probing (as opposed to a relaxation) subproblem of the corresponding node.
`Open Nodes`	Number of open nodes in branch-and-reduce tree.
`Time`	Current computational time in seconds. CPU time is reported for single-threaded jobs and wall clock time is reported for multi-threaded jobs.
`Lower Bound`	Current lower bound on the model.
`Upper Bound`	Current upper bound on the model.

Once the branch-and-reduce tree is searched, the best solution is isolated and a corresponding dual solution is calculated. Then, the total number of branch-and-reduce iterations (number of search tree nodes) is reported, followed by the node where the best solution was identified (a `-1` indicates preprocessing as explained in the next section on termination messages).

5.2 Termination messages, model and solver statuses

Upon normal termination, BARON will report the node where the optimal solution was found. We refer to this node as `nodeopt`. The log message is of the form:

```
Best solution found at node:  (nodeopt)
```

where `nodeopt` can take the following values:

$$\texttt{nodeopt} = \begin{cases} -3 & \text{no feasible solution found,} \\ -2 & \text{the best solution found was the user-supplied,} \\ -1 & \text{the best solution was found during preprocessing,} \\ i & \text{the best solution was found in the } i\text{th node of the tree.} \end{cases}$$

BARON, In addition to reporting `nodeopt`, will issue one of the following statements upon termination:

- `*** Normal completion ***`. This is the most desirable termination status. The problem has been solved within tolerances in this case. If BARON returns a code of `-3`, then no feasible solution exists.
- `*** Heuristic termination ***`. While global optimality is not guaranteed in this case, BARON will terminate with this message when (a) a feasible solution has been found and (b) the progress of lower/upper bounds satisfies the heuristic termination criterion set by the user through BARON's `DeltaTerm` option.
- `*** User did not provide appropriate variable bounds ***`. The user will need to read the BARON `summary` file for pointers to variables and expressions with missing bounds. The model should be modified in order to provide bounds for variables and intermediate expressions that make it possible for BARON to construct reliable relaxations. Even though relaxation bounds are printed on the screen to give the user a feeling for convergence, these bounds may not be valid for the problem at hand. This message is followed by one of the following two messages:

  - `*** Infeasibility is therefore not guaranteed ***`. This indicates that, because of missing bounds, no feasible solution was found but model infeasibility was not proven.
  - `*** Globality is therefore not guaranteed ***`. This indicates that, because of missing bounds, a feasible solution was found but global optimality was not proven.

- `*** Max. allowable nodes in memory reached ***`. The user will need to increase the physical memory of the computer or change algorithmic options, such as branching and node selection rules, to reduce the size of the search tree and memory required for storage.
- `*** Max. allowable BaR iterations reached ***`. The user will need to increase the maximum number of allowable iterations. The BARON option is `MaxIter`.
- `*** Max. allowable time exceeded ***`. The user will need to increase the maximum of allowable time. The BARON option is `MaxTime`.
- `*** Problem is numerically sensitive ***`. BARON is designed to automatically handle problems with numerical difficulties. However, for certain problems, the global optimum is numerically sensitive. This occurs, for instance, when the objective function value varies significantly over small neighborhoods of points that are strictly outside the feasible region but are nonetheless feasible within numerical tolerances. When BARON returns this message, the "Best possible" reported on the objective is likely correct.
- `*** Search interrupted by user ***`. The run was interrupted by the user (Ctrl-C).
- `*** Insufficient Memory for Data structures ***`. More memory is needed to set up the problem data structures. The user will need to increase the physical memory available on the computer in order to accommodate problems of this size.
- `*** A potentially catastrophic access violation just took place`. In the unlikely event of an access violation, BARON will terminate the search and return the best known solution. Please report problems that lead to this termination condition to Nick Sahinidis (`niksah@minlp.com`).

5.3 BARON solution output

When BARON is used under AIMMS, AMPL, GAMS, JuMP, MATLAB, Pyomo, or YALMIP, the corresponding BARON interface brings BARON results into these modeling environments. Therefore, users of these systems can skip this section. For users who choose to use BARON outside of these modeling systems, BARON's solution must be read from three output files:

- The `results` file provides the results. Each solution found by BARON is reported in this file as soon as it is found. Variable values and dual values for variables and constraints are printed in the order in which variables and constraints are defined in the BARON file. At the end of this file, a termination message, such as "*** Normal Completion ***" is printed, followed by the best solution point in two different formats, the last of which makes use of the variable names used in the BARON file.

- The `summary` file contains the information that goes to the screen. In addition, it provides information on missing bounds, if any.

- The `time` file contains a single line with concise information on the solution, including a breakdown of iterations and times (the same information is available at the bottom of the `summary` file as well.)

As detailed in Section 7, the user has full control on whether any of these files will be written or not. In addition, the user can specify the names and/or paths of these output files. The time file should be read first after BARON's termination in order to obtain information regarding termination status. This file contains a single line with the following space-separated items:

- `ProName`.
- The number of constraints of the optimization problem.
- The number of variables of the optimization problem.
- The number of constraints in one of BARON's core reformulations of the optimization problem.
- The number of variables in one of BARON's core reformulations of the optimization problem.
- BARON's lower bound for the global optimum of the problem.
- BARON's upper bound for the global optimum of the problem.
- BARON's solver status, which can take one of the following values:
  1. If normal completion occurred, i.e., the problem was solved within tolerances.
  2. If there is insufficient memory to store the number of nodes required for this search tree (increase physical memory or change algorithmic options).
  3. If the maximum allowed number of iterations was exceeded (increase `maxiter`).
  4. If the maximum allowed time was exceeded (increase `maxtime`).
  5. If the problem is numerically sensitive.
  6. If the run was interrupted by user (Ctrl-C)
  7. If there was insufficient memory to setup BARON's data structures (increase physical memory).
  8. This return code is reserved for development purposes.
  9. If the run was terminated by BARON.
  10. If the run was terminated by BARON's parser because of a syntax error in the BARON input file.
  11. If the run was terminated because of a licensing error.
  12. If the heuristic termination rule was invoked by the user.
- BARON's model status which can take one of the following values:
  1. optimal within tolerances

2. infeasible
3. unbounded
4. intermediate feasible
5. unknown

- If model status is 4 or 5, this entry denotes the number of missing bounds from variables/expressions that make BARON unable to guarantee global optimality.
- The number of branch-and-bound iterations taken
- The node where the best solution was found (`nodeopt`).
- The maximum number of nodes stored in memory.
- The total CPU time in seconds.
- The total wall clock time in seconds.

If `nodeopt` $= -3$, there will be no solution in the `results` file. Otherwise, the solution can be found in the `results` file by starting from the end of the file, searching backward for "***" and then reading the solution forward, one variable at a time. The variables are ordered in the way they were defined in the `VARIABLES` section of the BARON file. If available, the dual solution is also provided there. In addition, the best primal solution is provided using variable names. If the solution process is interrupted, for instance by Ctrl-C, the primal solution will be present in the `results` file but not necessarily the corresponding dual.

If BARON declares the problem as unbounded, it will report its best solution found, possibly followed by a vertex and direction of an unbounded ray at the end of the `results` file.

In the case of `numsol` > 1, BARON returns the best `numsol` solutions found. These solutions follow right after the "***" mentioned above and are sorted from worst to best. Duals may not be returned for all these solutions. For those solutions for which a corresponding dual was found, the dual is also printed right after the primal. There will typically be a dual solution for the best solution found and all local minima. However, there will be no dual for non-KKT points, something that is highly likely to happen in most applications.

When `numsol` $= -1$ (find all feasible solutions), the solutions are reported in the `results` file as soon as they are found. These solutions are reported before the "***" and can be read from this file by searching for occurrences of "found", reading the solution reported immediately thereafter, and repeating this process until all occurrences of "found" are identified. Again, many of these solutions will be reported without corresponding duals. At the end of the file, i.e., following "*** Succ...", the best solution can be read, along with a corresponding dual.

6 Some BARON features

The features described in this section rely on options that are further detailed in the next section. For details of the algorithmic implementations, the user may wish to consult publications cited at the end of this document.

6.1 No starting point is required

In contrast to many NLP algorithms that require a feasible starting point, a starting point is not required for BARON. A user may optionally provide a starting point for all or even some of the problem variables. BARON will judiciously initialize any variables that are not initialized by the user. Even when the problem functions cannot be evaluated at a user-provided starting point, BARON is still capable of performing its global search.

6.2 Finding a few of the best or all feasible solutions

BARON offers a facility through its `NumSol` option to find the best few, or even all feasible, solutions to a model. This facility is applicable to combinatorial as well as continuous problems. Even for the case of combinatorial problems, BARON does not rely on integer cuts to find multiple solutions. It instead utilizes a single search tree, thus providing a computationally efficient method for finding multiple solutions. Furthermore, because BARON's approach applies to integer as well as continuous programs, it can be used to find *all* feasible solutions to a system of nonlinear equality and inequality constraints. Note, however, that using this feature will make the branch-and-bound search much slower than normal.

Once a model is solved by BARON with the `NumSol` option, the solutions found can be read from BARON results file. To illustrate this feature, we consider a problem in kinematic analysis of robot manipulators, the so-called *indirect-position* or *inverse kinematics* problem, in which the desired position and orientation of a robot hand is given and the relative robot joint displacements are to be found. The specific example that we consider involves the following set of equations for the PUMA robot:

$$
\begin{aligned}
&\gamma_1 x_1 x_3 + \gamma_2 x_2 x_3 + \gamma_3 x_1 + \gamma_4 x_2 + \gamma_5 x_4 + \gamma_6 x_7 + \gamma_7 = 0\\
&\gamma_8 x_1 x_3 + \gamma_9 x_2 x_3 + \gamma_{10} x_1 + \gamma_{11} x_2 + \gamma_{12} x_4 + \gamma_{13} = 0\\
&\gamma_{14} x_6 x_8 + \gamma_{15} x_1 + \gamma_{16} x_2 = 0\\
&\gamma_{17} x_1 + \gamma_{18} x_2 + \gamma_{19} = 0\\
&x_1^2 + x_2^2 - 1 = 0\\
&x_3^2 + x_4^2 - 1 = 0\\
&x_5^2 + x_6^2 - 1 = 0\\
&x_7^2 + x_8^2 - 1 = 0\\
&-1 \le x_i \le 1, \quad i = 1, \ldots, 8
\end{aligned}
$$

where

$$
\begin{array}{llll}
\gamma_1 = 0.004731 & \gamma_6 = 1 & \gamma_{11} = -0.07745 & \gamma_{16} = 0.004731\\
\gamma_2 = -0.3578 & \gamma_7 = -0.3571 & \gamma_{12} = -0.6734 & \gamma_{17} = -0.7623\\
\gamma_3 = -0.1238 & \gamma_8 = 0.2238 & \gamma_{13} = -0.6022 & \gamma_{18} = 0.2238\\
\gamma_4 = -001637 & \gamma_9 = 0.7638 & \gamma_{14} = 1 & \gamma_{19} = 0.3461\\
\gamma_5 = -0.9338 & \gamma_{10} = 0.2638 & \gamma_{15} = 0.3578 &
\end{array}
$$

The first four equations of this problem are bilinear while the last four are generalized cylinders. BARON's scheme for finding all feasible solutions works well in continuous spaces as long as the

solutions are isolated (separated by a certain distance). The BARON option `isoltol` (default value of 10^{-4}) allows the user to specify the isolation tolerance used for discriminating among different solutions. In order for two feasible solution vectors to be considered different, at least one of their coordinates must differ by `isoltol`.

The BARON file for the robot problem is as follows:

```
//  Filename: robot.bar
//
//  Purpose: Find all solutions of the PUMA robot problem
//  L.-W. Tsai and A. P. Morgan, "Solving the kinematics of the
//  most general six- and five-degree-of-freedom manipulators by
//  continuation methods," ASME J. Mech. Transm. Automa. Des.,
//  107, 189-200, 1985.

OPTIONS{
numsol: 20;
}

VARIABLES  x1,x2,x3,x4,x5,x6,x7,x8;

LOWER_BOUNDS{
x1: -1;
x2: -1;
x3: -1;
x4: -1;
x5: -1;
x6: -1;
x7: -1;
x8: -1;
}

UPPER_BOUNDS{
x1: 1;
x2: 1;
x3: 1;
x4: 1;
x5: 1;
x6: 1;
x7: 1;
x8: 1;
}

EQUATIONS  e2,e3,e4,e5,e6,e7,e8,e9,e10,e11,e12,e13,e14,e15,e16;


e2: 0.004731*x1*x3 - 0.1238*x1 - 0.3578*x2*x3 - 0.001637*x2 - 0.9338*x4 + x7
```

```
        <= 0.3571;

e3: 0.1238*x1 - 0.004731*x1*x3 + 0.3578*x2*x3 + 0.001637*x2 + 0.9338*x4 - x7
        <= -0.3571;

e4: 0.2238*x1*x3 + 0.2638*x1 + 0.7623*x2*x3 - 0.07745*x2 - 0.6734*x4 - x7
        <= 0.6022;

e5: (-0.2238*x1*x3) - 0.2638*x1 - 0.7623*x2*x3 + 0.07745*x2 + 0.6734*x4 + x7
        <= -0.6022;

e6: x6*x8 + 0.3578*x1 + 0.004731*x2  <= 0;

e7:  - x6*x8 - 0.3578*x1 - 0.004731*x2  <= 0;

e8:  - 0.7623*x1 + 0.2238*x2 == -0.3461;

e9: x1^2 + x2^2  <= 1;

e10: (-x1^2) - x2^2  <= -1;

e11: x3^2 + x4^2  <= 1;

e12: (-x3^2) - x4^2  <= -1;

e13: x5^2 + x6^2  <= 1;

e14: (-x5^2) - x6^2  <= -1;

e15: x7^2 + x8^2  <= 1;

e16: (-x7^2) - x8^2  <= -1;

OBJ: minimize    0;
```

The above problem has 14 different solutions. Looking at the BARON results file, these solutions can be found after the "*** Normal Completion ***" message.

6.3 Using BARON as a multi-start heuristic solver

To gain insight into the difficulty of a nonlinear program, especially with regard to the existence of multiple local solutions, modelers often make use of multiple local searches from randomly generated starting points. This can easily be accomplished with BARON's `NumLoc` option, which determines the number of local searches to be performed by BARON's preprocessor. BARON can be forced to terminate after preprocessing by setting the number of iterations to 0 through the `MaxIter` option. In addition to local search, BARON's preprocessor performs extensive reduction

of variable ranges. To sample the search space for local minima without range reduction, the user would have to set the range reduction options `TDo, MDo, LBTTDo`, and `OBTTDo` to zero. On the other hand, leaving these options to their default values increases the likelihood of finding high quality local optima during preprocessing. If `NumLoc` is set to -1, local searches in preprocessing will be done from randomly generated starting points until global optimality is proved or `MaxTime` seconds have elapsed.

6.4 Systematic treatment of unbounded problems

If BARON declares a problem as unbounded, it will search for and may report a vertex and direction of an unbounded ray. In addition, BARON will report the best solution found. This will be a feasible point that is as far along as possible on an unbounded ray while avoiding numerical errors because of floating point arithmetic.

6.5 Systematic treatment of infeasible problems

If BARON declares a problem as infeasible, it has the capability to identify a subset of the constraints that are infeasible and become feasible once any one of them is eliminated. This, so-called, *irreducibly inconsistent system* (IIS) can be obtained by BARON for all types of problems handled by BARON, including linear and nonlinear, continuous and integer, convex and nonconvex, and problems with complementarity constraints. BARON's `CompIIS` option can be used to identify an IIS.

As an example, consider the problem of minimizing the nonconvex function x_1x_3 over the following nonconvex constrained set:

$$\begin{aligned}
\text{e1}: &\quad 85 + 0.006x_2x_5 + 0.0006x_1x_4 - 0.002x_3x_5 <= 92\\
\text{e2}: &\quad 0.8x_2x_5 + 0.003x_1x_2 + 0.002x_3^2 = 110\\
\text{e3}: &\quad 9 + 0.005x_3x_5 + 0.001x_1x_3 + 0.002x_3x_4 <= 25\\
&\quad 78 \le x_1 \le 102\\
&\quad 33 \le x_2 \le 45\\
&\quad 27 \le x_i \le 45, \quad i = 3, \ldots, 5
\end{aligned}$$

When this problem is solved with `CompIIS` equal to 1, BARON provides the following infeasible set in the results file:

```
IIS contains 1 row and 3 columns as follows:
 e2   Upper
 x1   Lower
 x2   Lower
 x5   Lower
```

The IIS consists of the lower bounds of variables x_1, x_2, and x_5, along with the $\le$ part of the equality constraint e2. This suggests that constraint e2 and the entire model can be made feasible by lowering the lower bound of any of the three variables that are part of the IIS, whereas modifying the bounds of x_3 would not make the model feasible.

If a problem is known to be infeasible and the user desires to identify an IIS, it may be beneficial to set BARON's `NumLoc` option to zero. Doing so will deactivate BARON's initial upper bounding search, which involves multiple local searches. A nonzero value of `DoLocal` is still desired in order to permit local search during the solution of certain subproblems that BARON solves while searching for an IIS. Identification of an IIS requires BARON to turn off some of its presolve facilities. As a result, activating IIS detection for a feasible model may lead to performance degradation.

6.6 Handling of complementarity constraints

Complementarity relationships of the type $f(x)g(x) = 0$ are automatically recognized and exploited algorithmically by BARON. The functions f and g may be univariate or multivariate, linear or nonlinear, convex or nonconvex, in terms of continuous and/or integer variables, and may be subject to additional constraints in the model. These complementarity relationships can be inferred by BARON even when implied by problem constraints and variable bounds. As a result, BARON can solve general mathematical programs with equilibrium constraints (MPECs). This class of problems includes the classical linear complementarity problem

$$\text{(LCP):}\quad \text{Find } z \geq 0 \text{ and } q \text{ such that } Mz + q \geq 0 \text{ and } z^t(Mz + q) = 0$$

as well as the more general mixed complementarity problem

$$\begin{aligned}\text{(MCP):}\quad & \text{Given a function } f : \mathbb{R}^n \to \mathbb{R}^n \text{ and bounds } l, u \in \overline{\mathbb{R}}^n \text{ with} \\ & \overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, +\infty\},\ \text{find } z \in \mathbb{R}^n \text{ and } w, v \in \mathbb{R}^n_+ \text{ such that} \\ & f(z) = w - v,\ l \leq z \leq u,\ (z - l)^t w = 0,\ (u - z)^t v = 0\end{aligned}$$

Both problems are automatically recognized and exploited by BARON without the user having to mark complementarities in any special way.

6.7 Parallel capabilities

For difficult problems with integer variables, most of BARON's time is spent on solving MIP relaxations. Thus, considerable speedups may be obtained via parallel solution of the MIP subproblems. For this purpose, the option `threads` may be used to specify the number of cores that BARON's MIP subsolver is allowed to use. By default, this option has the value of 1, meaning that a single core will be utilized.

7 The BARON options

The BARON options allow the user to control termination tolerances, branching and relaxation strategies, heuristic local search options, and output options as detailed in this section.

Contrary to variable names, the BARON parser is not case-sensitive to option names.

7.1 Termination options

Option	Description	Default
`EpsA` (ϵ_a)	Absolute termination tolerance. BARON terminates if $\|U - L\| \leq \epsilon_a$, where U and L are the values of the incumbent and best estimate, respectively, for the optimization problem at the current iteration. `EpsA` must be a real greater than or equal to 1e-12.	`1e-6`
`EpsR` (ϵ_r)	Relative termination tolerance. BARON terminates if $\|U - L\| \leq \epsilon_r \|U\|$, where U and L are the values of the incumbent and best estimate, respectively, for the optimization problem at the current iteration. `EpsR` must be a nonnegative real.	`1e-9`
`DeltaTerm`	Users have the option to request BARON to terminate if insufficient progress is made over δ_t consecutive seconds. Progress is measured using the absolute and relative improvement thresholds δ_a and δ_r defined below. Termination will occur if, over a period of δ_t consecutive seconds, the value of the best solution found by BARON is not improved by at least an absolute amount δ_a or an amount equal to δ_r times the value of the incumbent at time $t - \delta_t$. This termination condition is enforced after processing the root node and only after a feasible solution has been obtained. Because it relies on CPU time measurements, which may depend on machine load, this option may result in nondeterministic behavior. `0:` do not enforce this termination condition `1:` terminate if progress is insufficient	`0`
`DeltaT` (δ_t)	If `DeltaTerm` is set to 1, BARON will terminate if insufficient progress is made over δ_t consecutive seconds. If δ_t is set to a non-positive quantity, BARON will automatically set δ_t equal to $-\delta_t$ times the CPU time taken till the end of root node processing. `DeltaT` can take any real value.	`-100`
`DeltaA` (δ_a)	Absolute improvement termination threshold. `DeltaA` must be a real greater than or equal to 1e-12.	∞
`DeltaR` (δ_r)	Relative improvement termination threshold. `DeltaR` must be a real greater than or equal to 1e-12.	1
`CutOff`	BARON will ignore parts of the search space that contain solutions with a worse objective function value than this value. `CutOff` can take any real value. The default value for this option is ∞ for minimization problems and $-\infty$ for maximization problems.	$\pm\infty$
`Target`	BARON may terminate as soon as a solution is identified that is at least as good as this value. `Target` can take any real value. The default value for this option is $-\infty$ for minimization problems and ∞ for maximization problems.	$\pm\infty$

`AbsConFeasTol`	Absolute constraint feasibility tolerance. This tolerance is used for general constraints and variable bounds. A point is considered feasible for a constraint/bound if the absolute or relative constraint feasibility tolerance is satisfied for each (bound) constraint. `AbsConFeasTol` must be a real greater than or equal to 1e-12.	`1e-5`
`RelConFeasTol`	Relative constraint feasibility tolerance. This tolerance is used for general constraints and variable bounds. A point is considered feasible for a constraint/bound if the absolute or relative constraint feasibility tolerance is satisfied for each (bound) constraint. `RelConFeasTol` must be a real between 0 and 0.1.	`0`
`AbsIntFeasTol`	Absolute integer feasibility tolerance. All integer variable values must satisfy this tolerance. A point is considered integer feasible for a variable if integrality is satisfied using the absolute or relative integer feasibility tolerance. `AbsIntFeasTol` must be a real greater than or equal to 1e-12.	`1e-5`
`RelIntFeasTol`	Relative integer feasibility tolerance. All integer variable values must satisfy this tolerance. A point is considered integer feasible for a variable if integrality is satisfied using the absolute or relative integer feasibility tolerance. `RelIntFeasTol` must be a real between 0 and 0.1	`0`
`BoxTol`	Boxes will be eliminated if smaller than this tolerance. `BoxTol` must be a real greater than or equal to 1e-12.	`1e-8`
`FirstFeas`	If set to 1, BARON will terminate once it finds `NumSol` feasible solutions, irrespective of solution quality. By default, `FirstFeas` is 0, meaning that BARON will search for the *best* `NumSol` feasible solutions. `0:` do not enforce this termination condition `1:` terminate as soon as `NumSol` feasible solutions are found	`0`
`FirstLoc`	If set to 1, BARON will terminate once it finds a local optimum, irrespective of solution quality. By default, `FirstLoc` is 0, meaning that BARON will search for the *best* `NumSol` feasible solutions. Note that, when this option is set to 1, termination due to optimality tolerances may result in termination without a local minimum. `0:` do not enforce this termination condition `1:` terminate as soon as a local optimum is found	`0`

`MaxIter`	Maximum number of branch-and-reduce iterations allowed. -1 implies unlimited. Setting `MaxIter` to 0 will force BARON to terminate after root node preprocessing. Setting `MaxIter` to 1 will result in termination after the solution of the root node. `MaxIter` must be an integer greater than or equal to -1.	`-1`
`MaxTime`	Maximum time allowed (sec). For single-threaded jobs, i.e., when `threads` equals 1, this limit is enforced on CPU time consumed by the job. For multi-threaded jobs, the `MaxTime` limit is enforced on wall clock time. Setting `MaxTime` to -1 will make BARON ignore the time limit. `MaxTime` must be a real equal to -1 or greater than 0.	`1000`
`WantDual`	If set to 1, BARON will return a dual solution. BARON uses an inexpensive technique to solve a KKT system for finding a dual solution corresponding to the best primal solution identified. If `WantDual` is set to 0, BARON may or may not return a dual solution.	`1`
`NumSol`	Number of feasible solutions to be found. By default, only one solution is sought. If this option is utilized, multiple solutions can be found at the expense of (much) higher CPU time. Solutions found will be listed in the `results` file. As long as `NumSol` $\neq$ `-1`, these solutions will be sorted from best to worse. If `NumSol` is set to -1, BARON will search for all feasible solutions to the given model and print them, in the order in which they are found, in the `results` file. `NumSol` must be an integer equal to -1 or greater than or equal to 1.	`1`
`IsolTol`	Separation distance between solutions. This option is used in conjunction with `NumSol`. For combinatorial optimization problems, feasible solutions are isolated. For continuous problems, feasible solution points within an l_∞ distance that does not exceed `IsolTol` $>$ `0` will be treated as identical by BARON. `IsolTol` can take any positive value greater than or equal to 1e-12.	`1e-4`

7.2 Relaxation options

Option	Description	Default
`NOuter1`	Number of outer approximators of convex univariate functions. `NOuter1` must be a nonnegative integer.	`4`
`NOutPerVar`	Number of outer approximators per variable for convex multivariate functions. `NOutPerVar` must be a nonnegative integer.	`4`

`NOutIter`	Number of rounds of cutting plane generation at node relaxation. `NOutIter` must be a nonnegative integer.	`4`
`OutGrid`	Number of grid points per variable for convex multivariate approximators of BARON's `CONVEX_EQUATIONS`. `OutGrid` must be a nonnegative integer.	`20`

7.3 Range reduction options

Option	Description	Default
`TDo`	Nonlinear-feasibility-based range reduction option (poor man's NLPs). `0:` no bounds tightening is performed `1:` bounds tightening is performed	`1`
`MDo`	Marginals-based reduction option. `0:` no range reduction based on marginals `1:` range reduction done based on marginals	`1`
`LBTTDo`	Linear-feasibility-based range reduction option (poor man's LPs). `0:` no range reduction based on feasibility `1:` range reduction done based on feasibility	`1`
`OBTTDo`	Optimality-based tightening option. `0:` no range reduction based on optimality `1:` range reduction done based on optimality	`1`
`PDo`	Number of probing problems allowed. `-2:` automatically decided by BARON `0:` no range reduction by probing `-1:` probing on all variables n`:` probing on n variables	`-2`

7.4 Tree management options

Option	Description	Default
`BrVarStra`	Branching variable selection strategy. `0:` BARON's dynamic strategy `1:` largest violation `2:` longest edge	`0`
`BrPtStra`	Branching point selection strategy. `0:` BARON's dynamic strategy `1:` ω-branching `2:` bisection-branching `3:` convex combination of ω and bisection	`0`

NodeSel	Specifies the node selection rule to be used for exploring the search tree. 0: BARON's strategy 1: best bound 2: LIFO 3: minimum infeasibilities	0

7.5 Local search options

Option	Description	Default
DoLocal	Local search option for upper bounding. 0: no local search is done during upper bounding 1: BARON automatically decides when to apply local search based on analyzing the results of previous local searches	1
NumLoc	Number of local searches done in preprocessing. The first one begins with the user-specified starting point. Subsequent local searches are done from judiciously chosen starting points. If NumLoc is set to -1, local searches in preprocessing will be done until proof of globality or MaxTime is reached. If NumLoc is set to -2, BARON decides the number of local searches in preprocessing based on problem and NLP solver characteristics. NumLoc must be an integer greater than or equal to -2.	-2

7.6 Output and file name options

During run time, BARON utilizes a number of files. The generation of some of them is optional. In all cases, file names can be controlled by the user. File names must be unique for each BARON run in case of parallel runs in the same execute directory.

Option	Description	Default
PrFreq	Log output frequency in number of nodes.	1000000
PrTimeFreq	Log output frequency in number of seconds.	30
PrLevel	Option to control log output. 0: all log output is suppressed 1: print log output	1
LocRes	Option to control output to log from local search. 0: no local search output 1: detailed results from local search will be printed to the results file	0
ProName	Problem name. This option must be provided in double quotes and be no longer than 10 characters.	problem

`results`	Indicator if a `results` file is to be created. `0:` do not create file `1:` create file named according to the `ResName` option	`1`
`ResName`	Name of `results` file to be written. This option must be provided in double quotes in the `.bar` file.	`res.lst`
`summary`	Indicator if a `summary` file is to be created. `0:` do not create file `1:` create file named according to the `SumName` option.	`0`
`SumName`	Name of `summary` file to be written. This option must be provided in double quotes in the `.bar` file.	`sum.lst`
`times`	Indicator if a `times` file is to be created. `0:` do not create file `1:` create file named according to the `TimName` option.	`0`
`TimName`	Name of `times` file to be written. This option must be provided in double quotes in the `.bar` file.	`tim.lst`
`OptName`	Name of `options` file to be written (this file is for BARON usage only during run time and is not returned to the user). This option must be provided in double quotes in the `.bar` file.	`options`

7.7 Subsolver options

Option	Description	Default
`LPSol`	Specifies the LP/MIP solver to be used. By default, BARON will select the LP solver and may switch between different LP solvers during the search according to problem characteristics and solver performance. The solvers CPLEX, if available, and CBC will be used for this purpose. A single specific LP solver can be specified by setting this option to a value other than the default. If the specified solver is not licensed, BARON will default to automatic solver selection. `-1:` automatic LP solver selection `3:` CPLEX `8:` CLP/CBC `15:` HSL's LA04	`-1`
`AllowCPLEX`	In case of automatic LP/MIP solver selection, this option can be used to selectively permit or disallow the use of CPLEX as an LP/MIP subsolver. `0:` do not use CPLEX for LP/MIP subproblems `1:` consider CPLEX for LP/MIP subproblems	`1`

AllowCBC	In case of automatic LP/MIP solver selection, this option can be used to selectively permit or disallow the use of CBC as an LP/MIP subsolver. 0: do not use CBC for LP/MIP subproblems 1: consider CBC for LP/MIP subproblems	1
AllowHSL	In case of automatic LP/MIP solver selection, this option can be used to selectively permit or disallow the use of HSL's LA04 as an LP/MIP subsolver. 0: do not use HSL's LA04 for LP/MIP subproblems 1: consider HSL's LA04 for LP/MIP subproblems	1
CplexLibName	If utilized, this option must be supplied in double quotes and provide the entire path to the location of the CPLEX callable libraries on the user's computer. If left unspecified and LPSol is 3, BARON will utilize standard library location facilities to locate CPLEX and use it for the solution of LP/NLP subproblems. In the latter case, the CPLEX libraries should be in the user's LIBRARY PATH. When searching for the libraries on Windows systems, BARON will look for cplex12100.dll. On Linux systems, it will look for libcplex.so, and on MAC OSX, it will look for libcplex.dylib. If a CPLEX library named CplexLibName is not found, BARON will search for alternate versions (currently cplex1290.dll, libcplex12100.so or libcplex12100.dylib, depending on the platform). If CPLEX is still not found, BARON will resort to using CLP/CBC instead. As an alternative to the CplexLibName option, users of the stand-alone BARON code may copy the CPLEX libraries or use symbolic links to place the default library name on their user LIBRARY PATH. On Unix systems, the environment variable that controls the LIBRARY PATH is specified by $LD_LIBRARY_PATH; on OSX, it is $DYLD_LIBRARY_PATH. The CplexLibName option is not applicable to BARON under AIMMS, AMPL, or GAMS.	libcplex.so cplex12100.dll libcplex.dylib
LPAlg	Specifies the LP algorithm to be used. 0: automatic selection of LP algorithm 1: primal simplex 2: dual simplex 3: barrier	0

`NLPSol`	Specifies the NLP solver to be used. By default, BARON will select the NLP solver and may switch between different NLP solvers during the search according to problem characteristics and solver performance. Any combination of licensed NLP solvers may be used in that case. A single specific NLP solver can be specified by setting this option to a value other than the default. If the specified solver is not licensed, BARON will default to automatic solver selection. `-1:` automatic solver selection `0:` Local search based on function evaluations alone with no calls to local solvers `9:` IPOPT `10:` FilterSD `14:` FilterSQP	`-1`
`AllowFilterSD`	In case of automatic NLP solver selection, this option can be used to selectively permit or disallow the use of FilterSD as an NLP subsolver. `0:` do not use FilterSD for local search `1:` consider FilterSD for local search	`0`
`AllowFilterSQP`	In case of automatic NLP solver selection, this option can be used to selectively permit or disallow the use of FilterSQP as an NLP subsolver. `0:` do not use FilterSQP for local search `1:` consider FilterSQP for local search	`1`
`AllowIpopt`	In case of automatic NLP solver selection, this option can be used to selectively permit or disallow the use of IPOPT as an NLP subsolver. `0:` do not use IPOPT for local search `1:` consider IPOPT for local search	`1`

7.8 Licensing options

Option	Description	Default
`LicName`	License file name. If utilized, this option must be supplied in double quotes and provide the entire path to the location of the BARON license file. If left unspecified, BARON will search for the BARON license file `baronlice.txt` in the user PATH. This option is not applicable to BARON under AIMMS, AMPL, or GAMS.	`baronlice.txt`

7.9 Other options

Option	Description	Default
`CompIIS`	In case of an infeasible problem, this option can be used to search for an IIS. If this option is utilized, BARON prints the IIS it identifies in its `summary` file. Setting `CompIIS` equal to 1 works very well for most infeasible problems. Possible values are: `0:` do not search for an IIS `1:` the search for an IIS is based on a fast heuristic `2:` an IIS is obtained using a deletion filtering algorithm `3:` an IIS is obtained using an addition filtering algorithm `4:` an IIS is obtained using an addition-deletion filtering algorithm `5:` an IIS is obtained using a depth-first search algorithm	`0`
`IISint`	When search for an IIS is requested through `CompIIS`, BARON assumes that the model is unlikely to include an error in terms of binaries, i.e. the binary definitions are assumed correct and the IIS output should be interpreted with respect to binary definitions. General integer bounds may be assumed as correct or can be questioned using the option `IISint`, which can take the values of 0 (default, where integer bounds are assumed correct and the IIS should be interpreted with respect to integer bounds) or 1 (signalling that general integer bounds should be questioned). Integrality is enforced in both cases. `0:` do not consider general integers as part of an IIS `1:` consider general integers (but not binaries) as part of an IIS	`0`
`IISorder`	Defines the order in which problem constraints are considered in the search for an IIS. `-1:` auto set to aim for a small IIS depending on the value of `CompIIS` `1:` arrange constraints in problem order `2:` arrange constraints in ascending order of degree `3:` arrange constraints in descending order of degree `>=4:` random order using `IISorder` as seed	`-1`
`threads`	Specifies the number of cores that BARON is allowed to use for solution of its MIP subproblems. By default, this option has the value of 1, meaning that a single core will be utilized. The value of this option is passed to CBC, CPLEX, and XPRESS through the options -threads, CPX_PARAM_THREADS, and XPRS_THREADS, respectively.	`1`

`ProblemIsConvex`	If set to 1, this option tells BARON that the problem or its continuous relaxation is convex. `0:` do not assume that the continuous relaxation of the problem is convex `1:` assume that the continuous relaxation of the problem is convex	`0`
`seed`	Specifies the initial seed for BARON's random number generator. Changing the value of this option is likely to change the outcome of BARON's randomized search steps, including starting points and solutions obtained from local search heuristics. Must be a positive integer.	`19631963`

8 Bibliography

A partial listing of BARON-related publications that describe the algorithms implemented in the software, the theory behind them, and some related applications can be found at http://minlp.com/about-baron.