BARON user manual v. 13.0.0

April 17, 2014

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

Contents

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 guaranteed to provide global optima only 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 availability of finite lower and upper bounds on problem variables and their 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 at `http://www.minlp.com/download` and 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 MATLAB. In addition, BARON distributors GAMS and AIMMS provide licenses for using BARON under their modeling systems.

The software includes the Coin solvers CLP and IPOPT for solving BARON's linear programming (LP) and nonlinear programming (NLP) subproblems, respectively. Licensed LP and NLP solvers are optional and may expedite convergence. The latter are currently available only with the GAMS and AIMMS versions of BARON. Current valid LP subsolvers for BARON are CLP, CPLEX, and XPRESS. Current valid NLP subsolvers are CONOPT, IPOPT, MINOS, and SNOPT.

2 Model requirements

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

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

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. 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. GAMS/BARON and AIMMS/BARON automatically handle $|x|$ and x^y, where x and y are variables; otherwise, suitable transformations discussed below can be used. Currently, there is no support for other functions, including the trigonometric functions $\sin(x)$, $\cos(x)$, etc.

2.2 Variable and expression bounds

Nonlinear variables and 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 on as many problem variables as possible. However, providing finite bounds on variables alone may not suffice 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's preprocessor will attempt to infer appropriate bounds from problem constraints. If this step fails, global optimality of the solutions provided may not be guaranteed.

3 BARON input

There are various ways for passing an optimization problem to BARON:

- Directly, using the BARON modeling language.
- Indirectly, using one of the available BARON interfaces under Matlab, AIMMS, or GAMS.

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

3.1 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 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.

3.2 Input grammar

The following rules should be adhered to while preparing a BARON input file:

- All statements should be terminated by a semicolon (`;`).
- Reserved words must appear in uppercase letters.
- Variable names can be in lower or upper case. The parser is case sensitive, i.e., `X1` and `x1` are two different variables.
- Variable names should be no longer than 15 characters.
- Variable names must start with a letter.
- 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.
- "ˆ" 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 be different in different computing environments:
  - $x\hat{}y\hat{}z = (x\hat{}y)\hat{}z$ in GAMS, Matlab, and Excel.
  - $x\hat{}y\hat{}z = x\hat{}(y\hat{}z)$ in Fortran, AMPL, BARON, Mathematica
- 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 GAMS, AIMMS 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 GAMS, AIMMS or Matlab.
- Parentheses ("(" and ")") can be used in any meaningful combination with operations in mathematical expressions.

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

3.3 The options section

This section is optional. If used, it should be placed on top of the file. 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 6. Options not specified here take their default values. Instead of `OPTIONS`, the word `OPTION` can also be used.

3.4 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 x3, 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.

- `LOCAL_EQUATIONS <list equation names>;`

  Similar to the previous two in usage.

- **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;
  ```

  Any 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;
  }
  ```

3.5 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 case 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.

3.6 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.theoptimizationfirm.com/download`.

3.7 Other ways to access BARON

For information on how to access BARON under Matlab, see the BARON/MATLAB interface manual at http://www.theoptimizationfirm.com/products/barmat. For information on how to access BARON under GAMS or AIMMS, consult the corresponding web sites of these modeling languages.

4 BARON output

4.1 BARON screen output

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

```
===========================================================================
 BARON version 13.0.0. Built: LNX-64 Mon Jan 13 22:09:56 EST 2014

 If you use this software, please cite:
 Tawarmalani, M. and N. V. Sahinidis, A polyhedral
 branch-and-cut approach to global optimization,
 Mathematical Programming, 103(2), 225-249, 2005.

 BARON is a product of The Optimization Firm, LLC. http://www.minlp.com/
 Parts of the BARON software were created at the
 University of Illinois at Urbana-Champaign.
===========================================================================
 No BARON license file found in user PATH.  Continuing in demo mode.
 Model size is allowable within BARON demo size.
 This BARON run may utilize the following subsolver(s)
 For LP:  COIN LP
 For NLP: COIN IPOPT with MUMPS and METIS
===========================================================================
 Starting solution is feasible with a value of    36.1767610000
 Doing local search
 Solving bounding LP
 Starting multi-start local search
 Preprocessing found feasible solution with value  1.00058825498
 Done with local search
===========================================================================
  Iteration    Open nodes       Total time    Lower bound      Upper bound
          1             1        000:00:00     1.00000          36.1768
          1             1        000:00:00     1.00000          327.489
*         7             6        000:00:00     1.00000          11.6106
*         7             6        000:00:00     1.00000          12.1141
*         7             6        000:00:01     1.00000          11.6559
*        12             9        000:00:01     1.00000          1.11397
*        12             9        000:00:01     1.00000          1.16749
*        12             9        000:00:01     1.00000          1.15795
*        12             7        000:00:01     1.00000          2.40994
*        13             8        000:00:01     1.00000          1.01772
*        13             8        000:00:01     1.00000          1.00601
*        13             8        000:00:01     1.00000          1.00069
*        13             8        000:00:01     1.00000          1.03484
*        13             8        000:00:01     1.00000          1.00006
*        15             8        000:00:01     1.00000          1.00188
*        16             8        000:00:01     1.00000          1.05901
*        17             9        000:00:01     1.00000          1.03244
*        19             9        000:00:01     1.00000          1.01447
*        27            14        000:00:01     1.00000          1.02129
*        30            17        000:00:01     1.00000          1.01574
*        31            17        000:00:01     1.00000          1.02728
*        35            19        000:00:01     1.00000          1.00476
*        40            19        000:00:02     1.00000          1.00006
*        40            19        000:00:02     1.00000          1.00954
*        42            21        000:00:02     1.00000          1.00018
*        43            20        000:00:02     1.00000          1.00954
*        45            20        000:00:02     1.00000          1.00488
*        45            20        000:00:02     1.00000          1.00004
*        52            20        000:00:02     1.00000          1.01225
*        58            22        000:00:02     1.00000          1.00545
*        65            21        000:00:02     1.00000          1.00314
*        66            20        000:00:02     1.00000          1.00935
*        76            18        000:00:02     1.00000          1.00006
*        76            18        000:00:02     1.00000          1.00476
*        88            21        000:00:02     1.00000          1.00476
*        89            21        000:00:02     1.00000          1.00476
*        89            21        000:00:02     1.00000          1.00004
*       100            22        000:00:02     1.00000          1.00288
*       111            21        000:00:02     1.00000          1.00061
*       112            20        000:00:02     1.00000          1.00111
*       114            20        000:00:02     1.00000          1.00188
*       115            19        000:00:02     1.00000          1.00006
*       119            19        000:00:02     1.00000          1.00004
*       122            20        000:00:02     1.00000          1.00151
*       124            20        000:00:02     1.00000          1.00006
*       145            20        000:00:02     1.00000          1.00079
*       147            18        000:00:02     1.00000          1.00078
*       152            19        000:00:02     1.00000          1.00062
*       156            21        000:00:02     1.00000          1.00014
*       157            20        000:00:02     1.00000          1.00006
*       161            20        000:00:02     1.00000          1.00006
*       177            17        000:00:02     1.00000          1.00028
*      187             21         000:00:02     1.00000            1.00000
*      191             19         000:00:02     1.00000            1.00018
*      213             19         000:00:02     1.00000            1.00010
*      230             21         000:00:02     1.00000            1.00013
*      247             20         000:00:02     1.00000            1.00009
*      263             24         000:00:02     1.00000            1.00006
*      267             23         000:00:02     1.00000            1.00006
*      386             27         000:00:02     1.00000            1.00001
*      436             23         000:00:02     1.00000            1.00000
*      458             19         000:00:02     1.00000            1.00000
*     1252             22         000:00:03     1.00000            1.00000
      1342              0         000:00:03     1.00000            1.00000

 Cleaning up

                        *** Normal completion ***

  LP  subsolver time:        000:00:00,    in seconds:          0.27
  NLP subsolver time:        000:00:02,    in seconds:          2.01
  Cutting time:              000:00:00,    in seconds:          0.04
  All other time:            000:00:00,    in seconds:          0.20

  Total time elapsed:        000:00:03,    in seconds:          2.52
       on parsing:           000:00:00,    in seconds:          0.01
       on preprocessing:     000:00:00,    in seconds:          0.10
       on navigating:        000:00:00,    in seconds:          0.04
       on relaxed:           000:00:00,    in seconds:          0.34
       on local:             000:00:02,    in seconds:          1.92
       on tightening:        000:00:00,    in seconds:          0.03
       on marginals:         000:00:00,    in seconds:          0.00
       on probing:           000:00:00,    in seconds:          0.06
       on IIS detection:     000:00:00,    in seconds:          0.00

  Total no. of BaR iterations:    1342
  Best solution found at node:    1342
  Max. no. of nodes in memory:      35

  Cut generation statistics (number of cuts / CPU sec)
    Bilinear                          0         0.00
    LD-Envelopes                    402         0.01
    Multilinears                      0         0.00
    Convexity                      6413         0.01
    Integrality                       0         0.03

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

The solver first tests feasibility of the user-supplied starting point. This point is found to be feasible with an objective function value of 36.176761. BARON subsequently does its own search and, during preprocessing, finds a feasible solution with an objective of 1.00058825498. Then, the iteration log provides information every 1,000,000 branch-and-bound iterations and every 30 seconds. Additionally, information is printed at the end of the root node, whenever the value of the incumbent is improved by at least 10^{-5}, and at the end of the 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 CPU 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.
`Total Time`	Current elapsed resource time in seconds.
`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). Finally, some information is provided about the number and type of cutting planes generated during the search.

4.2 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:

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

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

- `*** Normal completion ***`. This is the 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.

- `*** 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 CPU time exceeded ***`. The user will need to increase the maximum of allowable CPU 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.
- `*** Search terminated by BARON ***`. This will happen in certain cases when the required variable bounds are not provided in the input model. The user will need to read the BARON output for likely pointers to variables and expressions with missing bounds and fix the formulation, or be content with the solution provided. In the latter case the solution may not be globally optimal.
- `*** 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@gmail.com`).

4.3 BARON solution output

When BARON is used under Matlab, GAMS or AIMMS, the corresponding BARON interface brings BARON results into these modeling environments; these users can skip this section. For those users who choose to use BARON outside of Matlab, GAMS or AIMMS, 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 calculated. 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 in the bottom of `summary` file as well.)

As detailed in Section 6, 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 information:

- `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 CPU 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).
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.

- 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.

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. 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 they are sorted in improving order; the last solution is the 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 most 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 its corresponding dual solution.

5 Some BARON features

The features described in this section rely on options that are further detailed in the next section. The user may also wish to consult the Tawarmalani-Sahinidis book[1] for more details on BARON features and illustrations of their use.

5.1 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 carrying out its global search.

5.2 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. The development of this facility was motivated by combinatorial optimization problems but the facility is applicable to continuous problems as well. Even for combinatorial problems, BARON does not rely on integer cuts to find multiple solutions. Instead, it 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.

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:

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

---

[1] Tawarmalani, M. and N. V. Sahinidis, *Convexification and Global Optimization in Continuous and Mixed-Integer Nonlinear Programming: Theory, Algorithms, Software, and Applications*, 504 pages, Kluwer Academic Publishers, Dordrecht, Vol. 65 in *Nonconvex Optimization And Its Applications* series, 2002.

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 to be used to discriminate 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, all these solutions can be found after the “*** Normal Completion ***” message.

5.3 Using BARON as a multi-start heuristic solver

To gain insight into the difficulty of a nonlinear program, especially with regard to existence of multiple local solutions, modelers often make use of multiple local searches from randomly generated starting points. This can be easily done with BARON's `NumLoc` option, which determines the number of local searches to be done 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, one would have to set to 0 the range reduction options `TDo, MDo, LPTTDo`, and `OBTTDo`. 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` CPU seconds have elapsed.

5.4 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 as possible along an unbounded ray while avoiding numerical errors due to floating point arithmetic.

6 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.

6.1 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 lower and upper bounds 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 $L > \infty$ and $U - L \leq \epsilon_r \lvert L \rvert$, where U and L are the lower and upper bounds 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. `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 may ignore parts of the search space that contain solutions that are no better than this value. `CutOff` can take any real value.	∞
`AbsConFeasTol`	Absolute constraint feasibility tolerance. `AbsConFeasTol` must be a real greater than or equal to 1e-12.	`1e-5`
`RelConFeasTol`	Relative constraint feasibility tolerance. `RelConFeasTol` must be a nonnegative real.	`0`
`AbsIntFeasTol`	Absolute integer feasibility tolerance. `AbsIntFeasTol` must be a real greater than or equal to 1e-12.	`1e-5`
`RelIntFeasTol`	Relative integer feasibility tolerance. `RelIntFeasTol` must be a nonnegative real.	`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`

`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 CPU time allowed (sec). Setting `MaxTime` to -1 will make BARON ignore the time limit. `MaxTime` must be a real greater equal to -1 or greater than 0.	`1000`
`NumSol`	Number of feasible solutions to be found. 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 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 solutions 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`

6.2 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. `OutGrid` must be a nonnegative integer.	20

6.3 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`

6.4 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`

`BPInt`	Branching in BARON takes place on integer and nonlinear continuous variables. Users can specify branching priorities for any discrete and continuous variables using the `BRANCHING_PRIORITIES` keyword in the BARON input file. Default branching priorities in BARON are 1 for all variables, thus placing equal emphasis on all integer and continuous variables. The option `BPInt` can be used to adjust the priorities of integer variables. Priorities of integer variables are multiplied by `BPInt`. Larger values of `BPInt` place higher emphasis on integer variables for branching. `BPInt` must be a non-negative real.	1

6.5 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

6.6 Output options

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.	`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.	`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.	`tim.lst`

6.7 Subsolver options

Option	**Description**	**Default**
`LPSol`	Specifies the LP solver to be used. `3:` CPLEX `8:` CLP	3
`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, based on 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 `2`: MINOS `4`: SNOPT `9`: IPOPT	`-1`
`AllowMinos`	In case of automatic NLP solver selection, this option can be used to selectively permit or disallow the use of MINOS as an NLP subsolver. `0`: do not use MINOS for local search `1`: consider MINOS for local search	`1`
`AllowSnopt`	In case of automatic NLP solver selection, this option can be used to selectively permit or disallow the use of SNOPT as an NLP subsolver. `0`: do not use SNOPT for local search `1`: consider SNOPT 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`

6.8 Licensing options

Option	**Description**	**Default**
`LicName`	License file name. This option must be provided in double quotes and provide the entire path to the location of the BARON license file. Alternatively, the user can place the BARON license file `baronlice.txt` anywhere in the user PATH. This option is not applicable to BARON under GAMS or AIMMS.	`baronlice.txt`

7 Bibliography

The following is a partial list of BARON-related publications that describe the algorithms implemented in the software, the theory behind them, and some related applications.

1. H. S. Ryoo and N. V. Sahinidis. Global optimization of nonconvex NLPs and MINLPs with applications in process design. *Computers & Chemical Engineering*, 19:551–566, 1995.
2. M. C. Dorneich and N. V. Sahinidis. Global optimization algorithms for chip layout and compaction. *Engineering Optimization*, 25:131–154, 1995.
3. H. S. Ryoo and N. V. Sahinidis. A branch-and-reduce approach to global optimization. *Journal of Global Optimization*, 8:107–139, 1996.
4. N. V. Sahinidis. BARON: A general purpose global optimization software package. *Journal of Global Optimization*, 8:201–205, 1996.
5. R. A. Gutierrez and N. V. Sahinidis. A branch-and-bound approach for machine selection in just-in-time manufacturing systems. *International Journal of Production Research*, 34:797–818, 1996.
6. M. L. Liu, N. V. Sahinidis, and J. P. Shectman. Planning of chemical process networks via global concave minimization. In I. E. Grossmann (ed.)*, Global Optimization in Engineering Design,* Kluwer Academic Publishers, Boston, MA, pages 195–230, 1996.
7. V. Ghildyal. Design and Development of a Global Optimization System. Master's thesis, Department of Mechanical & Industrial Engineering, University of Illinois, Urbana, IL, 1997.
8. J. G. VanAntwerp, R. D. Braatz, and N. V. Sahinidis. Globally optimal robust control for systems with nonlinear time-varying perturbations. *Computers & Chemical Engineering*, 21:S125–S130, 1997.
9. J. P. Shectman and N. V. Sahinidis. A finite algorithm for global minimization of separable concave programs. *Journal of Global Optimization*, 12:1–36, 1998.
10. J. G. VanAntwerp, R. D. Braatz, and N. V. Sahinidis. Globally optimal robust control. *J. Process Control*, 9:375–383, 1999.
11. N. V. Sahinidis and M. Tawarmalani. Applications of global optimization to process and molecular design. *Computers & Chemical Engineering*, 24:2157–2169, 2000.
12. V. Ghildyal and N. V. Sahinidis. Solving global optimization problems with BARON. In A. Migdalas, P. Pardalos, and P. Varbrand (eds.)*, From Local to Global Optimization. A Workshop on the Occasion of the 70th Birthday of Professor Hoang Tuy, Linköping, Sweden, Aug. 24-29, 1997,* Kluwer Academic Publishers, Boston, MA, pages 205–230, 2001.
13. H. S. Ryoo and N. V. Sahinidis. Analysis of bounds for multilinear functions. *Journal of Global Optimization*, 19:403–424, 2001.
14. M. Tawarmalani and N. V. Sahinidis. Semidefinite relaxations of fractional programs via novel techniques for constructing convex envelopes of nonlinear functions. *Journal of Global Optimization*, 20:137–158, 2001.

15. M. Tawarmalani and N. V. Sahinidis. Convex extensions and convex envelopes of l.s.c. functions. *Mathematical Programming*, 93:247–263, 2002.

16. M. Tawarmalani and N. V. Sahinidis. *Convexification and Global Optimization in Continuous and Mixed-Integer Nonlinear Programming: Theory, Algorithms, Software, and Applications*. Kluwer Academic Publishers, Dordrecht, 2002.

17. M. Tawarmalani, S. Ahmed, and N. V. Sahinidis. Global optimization of $0-1$ hyperbolic programs. *Journal of Global Optimization*, 24:385–417, 2002.

18. M. Tawarmalani, S. Ahmed, and N. V. Sahinidis. Product disaggregation and relaxations of mixed-integer rational programs. *Optimization and Engineering*, 3:281–303, 2002.

19. N. V. Sahinidis, M. Tawarmalani, and M. Yu. Design of alternative refrigerants via global optimization. *AIChE Journal*, 49:1761–1775, 2003.

20. H. S. Ryoo and N. V. Sahinidis. Global optimization of multiplicative programs. *Journal of Global Optimization*, 26:387–418, 2003.

21. N. V. Sahinidis. Global optimization and constraint satisfaction: The branch-and-reduce approach. In AC. Bliek, C. Jermann, and A. Neumaier (eds.)*, Global Optimization and Constraint Satisfaction,* Lecture Notes in Computer Science, Vol. 2861, Springer, Berlin, pages 1–16, 2003.

22. S. Ahmed, M. Tawarmalani, and N. V. Sahinidis. A finite branch-and-bound algorithm for two-stage stochastic integer programs. *Mathematical Programming*, 100:355–377, 2004.

23. M. Tawarmalani and N. V. Sahinidis. Global optimization of mixed-integer nonlinear programs: A theoretical and computational study. *Mathematical Programming*, 99:563–591, 2004.

24. N. V. Sahinidis and M. Tawarmalani. Accelerating branch-and-bound through a modeling language construct for relaxation-specific constraints. *Journal of Global Optimization*, 32:259–280, 2005.

25. M. Tawarmalani and N. V. Sahinidis. A polyhedral branch-and-cut approach to global optimization. *Mathematical Programming*, 103:225–249, 2005.

26. Y. Chang and N. V. Sahinidis. Global optimization in stabilizing controller design. *Journal of Global Optimization*, 38:509–526, 2007.

27. X. Bao, N. V. Sahinidis, and M. Tawarmalani. Multiterm polyhedral relaxations for nonconvex, quadratically-constrained quadratic programs. *Optimization Methods and Software*, 24:485–504, 2009.

28. L. M. Rios and N. V. Sahinidis. Portfolio optimization for wealth-dependent risk preferences. *Annals of Operations Research*, 177:63–90, 2010.

29. X. Bao, N. V. Sahinidis, and M. Tawarmalani. Semidefinite relaxations for quadratically constrained quadratic programming: A review and comparisons. *Mathematical Programming*, 129:129–157, 2011.

30. A. Khajavirad, J. J. Michalek and N. V. Sahinidis. Relaxations of factorable functions with convex-transformable intermediates. *Mathematical Programming*, DOI: 10.1007/s10107-012-0618-8, 2012.
31. A. Khajavirad and N. V. Sahinidis. Convex envelopes of products of convex and component-wise concave functions. *Journal of Global Optimization*, 52:391–409, 2012.
32. A. Khajavirad and N. V. Sahinidis. Convex envelopes generated from finitely many compact convex sets. *Mathematical Programming*, 137:371–408, 2013.
33. K. Zorn and N. V. Sahinidis. Global optimization of general nonconvex problems with intermediate bilinear substructures. *Optimization Methods and Software*, 29:442–462, 2013.
34. K. Zorn and N. V. Sahinidis. Computational experience with applications of bilinear cutting planes. *Industrial & Engineering Chemistry Research*, accepted for publication, 2013.