BARON

Nick Sahinidis; Carnegie Mellon University, Department of Chemical Engineering, 5000 Forbes Avenue, Pittsburgh, PA 15213, `sahinidis@cmu.edu`

Mohit Tawarmalani; Purdue University, Krannert School of Management, West Lafayette, IN 47907, `mtawarma@purdue.edu`

6 December 2011

Contents

1 Introduction

The Branch-And-Reduce Optimization Navigator (BARON) is a GAMS solver for the *global solution* of nonlinear (NLP) and mixed-integer nonlinear programs (MINLP).

While traditional NLP and MINLP algorithms are guaranteed to converge 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 include the availability of finite lower and upper bounds on the 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.

1.1 Licensing and software requirements

In order to use GAMS/BARON, users will need to have a GAMS/BARON license. The software includes the Coin solvers CLP and IPOPT for solving BARON's linear programming (LP) and nonlinear programming (NLP) subproblems, respectively. Licensed GAMS LP and NLP solvers are optional and usually expedite convergence. Current valid LP subsolvers are CLP, CPLEX, and XPRESS. Current valid NLP subsolvers are CONOPT, IPOPT, MINOS, and SNOPT.

By default, GAMS/BARON will attempt to use CPLEX as the LP solver and MINOS as the NLP solver. The user can use the options `LPSol` and `NLPSol` to specify another LP or NLP solver. If the user does not have licenses for the default or user-specified LP/NLP solvers, GAMS/BARON will switch to the CLP and IPOPT solvers, as needed. GAMS/BARON can be used without a local NLP solver by setting `DoLocal` $= 0$ and `NumLoc` $= 0$. See §4 on the BARON options.

1.2 Running GAMS/BARON

BARON is capable of solving models of the following types: LP, MIP, RMIP, NLP, DNLP, RMINLP, and MINLP. If BARON is not specified as the default solver for these models, it can be invoked by issuing the following command before the solve statement:

```
option xxx=baron;
```

where `xxx` stands for `LP`, `MIP`, `RMIP`, `NLP`, `DNLP`, `QCP`, `MIQCP`, `RMINLP`, or `MINLP`.

2 Model requirements

In order to achieve convergence to global optimality, additional model constraints may be required. The additional constraints may speed up solver solution time and increase the probability of success.

2.1 Variable and expression bounds

All nonlinear variables and expressions in the mathematical program to be solved must be bounded below and above by finite numbers. It is important that finite lower and upper bounds be provided by the user on all problem variables. Note that providing finite bounds on variables is not sufficient 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. 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 is not guaranteed. Occasionally, because of the lack of bounds no numerically stable lower bounding problems can be constructed, in which case BARON may terminate.

See §4 on how to specify variable bounds.

2.2 Allowable nonlinear functions

In addition to multiplication and division, GAMS/BARON can handle nonlinear functions that involve $\exp(x)$, $\ln(x)$, x^α for real α, β^x for real β, x^y, and $|x|$. Currently, there is no support for other functions, including the trigonometric functions $\sin(x)$, $\cos(x)$, etc.

3   BARON output

3.1   BARON log output

The log output below is obtained for the MINLP model `gear.gms` from the GAMS model library using a relative and absolute tolerance of 1e-5.

```
===============================================================================
 BARON version 10.1.0. Built: LNX-64 Sat Dec 3 17:23:13 EST 2011

 Reference:
 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.
===============================================================================
 This BARON run utilizes the following subsolvers
 LP  solver: ILOG CPLEX
 NLP solver: MINOS
===============================================================================
 Starting solution is feasible with a value of     0.361767610000D+02
 Doing local search
 Preprocessing found feasible solution with value 0.125706576060D+01
 Solving bounding LP
 Starting multi-start local search
 Preprocessing found feasible solution with value 0.100209253056D+01
 Done with local search
===============================================================================
 We have space for 597641 nodes in the tree (in 96 MB memory)
===============================================================================
   Iteration    Open Nodes        Total Time      Lower Bound        Upper Bound
           1             1       000:00:00      0.100000D+01      0.100209D+01
*          1             1       000:00:00      0.100000D+01      0.100117D+01
           1             1       000:00:00      0.100000D+01      0.100117D+01
*          6+            5       000:00:00      0.100000D+01      0.100006D+01
*         19            14       000:00:00      0.100000D+01      0.100004D+01
*         48             0       000:00:00      0.100000D+01      0.100000D+01
          48             0       000:00:00      0.100000D+01      0.100000D+01

 Cleaning up solution and calculating dual


                         *** Normal Completion ***


  LP  subsolver time:         000:00:00,    in seconds:         0.02
  NLP subsolver time:         000:00:00,    in seconds:         0.01
  All other time:             000:00:00,    in seconds:         0.03

  Total time elapsed:         000:00:00,    in seconds:         0.05
        on parsing:           000:00:00,    in seconds:         0.00
        on preprocessing:     000:00:00,    in seconds:         0.01
        on navigating:        000:00:00,    in seconds:         0.02
        on relaxed:           000:00:00,    in seconds:         0.02
        on local:             000:00:00,    in seconds:         0.00
```

```
      on tightening:         000:00:00,    in seconds:          0.00
      on marginals:          000:00:00,    in seconds:          0.00
      on probing:            000:00:00,    in seconds:          0.01

  Total no. of BaR iterations:      48
  Best solution found at node:      48
  Max. no. of nodes in memory:      21

  Cut generation statistics (number of cuts / CPU sec)
    Multilinears                    0           0.00
    Bilinear                        0           0.00
    Convexity                     594           0.00

 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 0.361767610000D+02. BARON subsequently does its own search and, eventually, finds a feasible solution with an objective of 0.100209253056D+01. It then reports that the supplied memory (default of 96 MB) provides enough space for storing up to 597641 branch-and-reduce nodes for this problem.

Then, the iteration log provides information every 1000 iterations or every 30 seconds, whichever comes first. Additionally, information is printed at the end of the root node, whenever a better feasible solution is found, and at the end of the search. A star (*) in the first position of a line indicates that a better feasible solution was found. 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
Itn. no.	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. Finally, 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.

3.2   Termination messages, model and solver statuses

Upon termination, BARON will report the node where the optimal solution was found. We refer to this node as `nodeopt`. Associated with this node is a return code indicating the status of the solution found at `nodeopt`. The return code is given in the log line:

```
  Best solution found at node:  (return code)
```

The return codes have the following interpretation:

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

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 output (in file `sum.dat` in the scratch directory, use GAMS parameter `keep=1` to prevent the automatic removal of this directory) for likely 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 possible for BARON to construct reliable relaxations. 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 make more memory available to BARON or change algorithmic options to reduce the size of the search tree and memory required for storage. The user can increase the amount of available memory by using the GAMS options `WorkFactor` or `WorkSpace`.

- `*** Max. Allowable BaR Iterations Reached ***`. The user will need to increase the maximum number of allowable iterations. The BARON option is `MaxIter`. To specify this in GAMS, one can use the `NodLim` option. We remark that the BARON option `MaxIter` overrides `NodLim`.

- `*** Max. Allowable CPU Time Exceeded ***`. The user will need to increase the maximum of allowable CPU time. The BARON option is `MaxTime`. To specify this in GAMS, one can use the `ResLim` option. We remark that the BARON option `MaxTime` overrides `ResLim`.

- `*** Numerical Difficulties Encountered ***`. This case should be reported to the developers.

- `*** 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 can increase the amount of available memory by using the GAMS options `WorkFactor` or `WorkSpace`.

- `*** 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.

4 Some BARON features

The features described in these 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.

4.1 No starting point is required

For problems for which GAMS compilation is aborted because the nonlinear functions cannot be evaluated at the starting point, the user can use the following commands before the `SOLVE` statement:

```
MaxExecError = 100000;
option sys12 = 1;
```

---

[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.

The first command asks GAMS to continue compilation for as many as `MaxExecError` execution errors. The `sys12` option will pass the model to the BARON despite the execution errors. Even though the starting point is bad in this case, BARON is capable of carrying out its global search.

4.2 Finding the best, second best, third best, etc. solution, 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. Modelers interested in finding multiple solutions of integer programs often use integer cuts. The integer program is solved to optimality, an integer cut is added to the model in order to make the previous solution infeasible, and the model is solved again to find the second best integer solution. This process can then be repeated to obtain the best several solutions or all the feasible solutions of an integer program. This approach requires the user to explicitly model the integer cuts as part of the GAMS model.

In addition to eliminating the need for coding of integer cuts by the user, BARON does not rely on integer cuts to find multiple solutions. Instead, BARON directly eliminates a single search tree, which leads to a computationally more efficient method for finding multiple solutions. Furthermore, BARON's approach applies to integer as well as continuous programs. Hence, it can also 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 recovered using the GAMS GDX facility. An example is provided below.

```
$eolcom !
$Ontext
  Purpose: demonstrate use of BARON option 'numsol' to obtain the best
  numsol solutions of an optimization problem in a single branch-and-bound
  search tree.

  The model solved here is a linear general integer problem with 18 feasible
  solutions.  BARON is run with a request to find up to 20 solutions.  The
  model solved is the same as the one solved in gamslib/icut.gms.
$Offtext

set i index of integer variables  / 1 * 4 /

variables x(i) variables
          z     objective variable

integer variable x;

x.lo(i) = 2; x.up(i) = 4; x.fx('2') = 3;   ! fix one variable
x.up('4') = 3;   ! only two values

equation obj obj definition;

* pick an objective function which will order the solutions

obj .. z =e= sum(i, power(10,card(i)-ord(i))*x(i));

model enum / all /;

* instruct BARON to return numsol solutions
$onecho > baron.opt
numsol 20
gdxout multsol
$offecho
```

```
enum.optfile=1; option mip=baron, limrow=0, limcol=0, optca=1e-5,
optcr=1e-5; solve enum minimizing z using mip;

* recover BARON solutions through GDX
set sol /multsol1*multsol100/; variables xsol(sol,i), zsol(sol);

execute 'gdxmerge multsol*.gdx >  %gams.scrdir%merge.%gams.scrext%';
execute_load 'merged.gdx', xsol=x, zsol=z;

option decimals=8;

display xsol.l, zsol.l;
```

4.3   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`, `OBTTDo,` and `PreLPDo`. 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 `MaxPreTime` CPU seconds have elapsed.

5   The BARON options

BARON works like other GAMS solvers, and many options can be set in the GAMS model. The most relevant GAMS options are `ResLim`, `NodLim`, `OptCA`, `OptCR`, `OptFile`, and `CutOff`. The `IterLim` option is not implemented as it refers to simplex iterations and BARON specifies nodes. To specify BARON iterations, the user can set the `MaxIter` option, which is equivalent to the GAMS option `NodLim`. A description of GAMS options can be found in Chapter "Using Solver Specific Options."

If you specify "<modelname>`.optfile = 1;`" before the `SOLVE` statement in your GAMS model, BARON will then look for and read an option file with the name *baron.opt* (see "Using Solver Specific Options" for general use of solver option files). The syntax for the BARON option file is

```
   optname value
```

with one option on each line. For example,

```
  * This is a typical GAMS/BARON options file.
  * We will rely on the default BARON options with
  * two exceptions.
  pdo 3
  prfreq 1000
```

Lines beginning with * are considered comments and ignored. The first option specifies that probing will be used to reduce the bounds of three variables at every node of the tree. The second option specifies that log output is to be printed every 100 nodes.

The BARON options allow the user to control variable bounds and priorities, termination tolerances, branching and relaxation strategies, heuristic local search options, and output options as detailed next.

5.1 Setting variable bounds and branching priorities

Variable Bounds. BARON requires bounded variables and expressions to guarantee global optimality. The best way to provide such bounds is for the modeler to supply physically meaningful bounds for all problem variables using the `.lo` and `.up` variable attributes in the GAMS file. Alternatively, bounds may be provided to BARON in the form of *solver bounds* that are not be part of the user's model. To specify such solver bounds for BARON, create a BARON solver option file as described above. For lower and upper variable bounds the syntax is:

(variable)`.lo` *(value)*

(variable)`.up` *(value)*

For example, suppose we have a GAMS declaration:

```
Set i /i1*i5/;
Set j /j2*j4/;
variable v(i,j);
```

Then, the BARON bounds in the *baron.opt* file can be specified by:

```
v.lo            0
v.up            1
v.lo('i1','j2') 0.25
v.up('i1',*)    0.5
```

We specify that all variables `v(i,j)` have lower bounds of `0` and upper bounds of `1`, except variables over set element `i1`, which have upper bound `0.5`. The variable over set element `i1` and `j2` has lower bound `0.25`. Note that variable bounds are assigned in a procedural fashion so that bounds assigned later overwrite previous bounds.

Consider also the following GAMS example for expression bounds:

```
v =E= log(z);
z =E= x-y;
```

where `x,y,v,z` are variables. In order to ensure feasibility, we must have `x>y`, guaranteeing `z>0`. We can specify expression bounds in BARON using the solver option file:

```
z.lo 0.00001
```

which is equivalent to specifying in GAMS that the expression `x-y =G= 0.00001` and thereby bounding `v`.

Variable Priorities. BARON implements branch-and-bound algorithms involving convex relaxations of the original problem. Branching takes place not only on discrete variables, but also on continuous ones that are nonlinear. Users can specify branching priorities for both discrete and continuous variables.

To specify variable branching priorities, one specifies

(variable)`.prior` *(value)*

in the *baron.opt* file, where *(value)* can be any positive (real) value. Default priorities are `1` for all variables, including continuous ones. The option `bpint` can be used to adjust the priorities of integer variables. Priorities of integer variables are multiplied by `bpint`. By default, this option has a value of 1, thus placing equal emphasis on integer and continuous variables.

BARON priorities are assigned in a manner such that a larger value implies a higher priority. In contrast, GAMS priorities are assigned in such a fashion that a larger value implies a lower priority. BARON and GAMS variable priorities are related by

$$BARON\ priority = 1/GAMS\ priority$$

5.2 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 to the optimization problem at the current iteration. This is equivalent to the GAMS option OptCA.	1e-9
EpsR (ϵ_r)	Relative termination tolerance. BARON terminates if $L > \infty$ and $U - L \leq \epsilon_r |L|$, where U and L are the lower and upper bounds to the optimization problem at the current iteration. This is equivalent to the GAMS option OptCR.	0.1
ConTol	Constraint satisfaction tolerance.	1e-5
BoxTol	Box elimination tolerance.	1e-8
IntTol	Integrality satisfaction tolerance.	1e-6
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
MaxIter	Maximum number of branch-and-reduce iterations allowed. -1 implies unlimited. This is equivalent to the GAMS option NodLim. 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.	-1
MaxPreTime	Maximum CPU time allowed (sec) to be spent in preprocessing. If set to -1, the MaxTime limit apply.	-1
MaxTime	Maximum CPU time allowed (sec). This is equivalent to the GAMS option ResLim. If unspecified, the GAMS resource limit is enforced.	1200
NumSol	Number of feasible solutions to be found. Solutions found will be listed in the res.dat file in the GAMS scratch directory, use GAMS parameter keep=1 to prevent the automatic removal of this directory. 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 res.dat.	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.	1e-4

5.3 Relaxation options

Option	Description	Default
BilRel	Number of rounds of cutting plane generation from simultaneous convexification of bilinear terms at LP relaxation.	4
CvxRel	Number of rounds of BARON's core cutting plane generation, including convexity exploitation, at LP relaxation.	4
linearidentify	Identification of common linear subexpressions of nonlinear functions is done by default during relaxation construction; it can be turned off by using a value of 0 for this option. The default value may result in tighter relaxations but some models may require a large time during BARON's parsing and reformulation stage when this option is in effect.	1
MultMSize	Size of maximum allowable multilinear function for cutting plane generation; larger multilinear functions are decomposed to multilinears of size no more than this parameter.	7
MultRel	Number of rounds of cutting plane generation from envelopes of multilinear functions at LP relaxation.	1
nOuter1	Number of outer approximators of convex univariate functions.	4
NOutPerVar	Number of outer approximators per variable for convex multivariate functions.	4
NLPDoLin	Linearization option for relaxation. A value of 0 will result in the use of nonlinear relaxations whenever possible. This option should be avoided. It is offered as an alternative for hard problems but may lead to incorrect results depending on the performance of the local search solver for the problem at hand. The default value of 1 is to use a linear programming relaxation, which represents the most reliable approach under BARON.	1
NOutIter	Number of rounds of cutting plane generation at LP relaxation.	4
OutGrid	Number of grid points per variable for convex multivariate approximators.	20

5.4 Range reduction options

Option	Description	Default
TDo	Nonlinear-feasibility-based range reduction option (poor man's NLPs).	1
	`0:` no bounds tightening is performed.	
	`1:` bounds tightening is performed.	
MDo	Marginals-based reduction option.	1
	`0:` no range reduction based on marginals.	
	`1:` range reduction done based on marginals.	
LBTTDo	Linear-feasibility-based range reduction option (poor man's LPs).	1
	`0:` no range reduction based on feasibility.	
	`1:` range reduction done based on feasibility.	
OBTTDo	Optimality-based tightening option.	1
	`0:` no range reduction based on optimality.	
	`1:` range reduction done based on optimality.	
PDo	Number of probing problems allowed.	3
	`0:` no range reduction by probing.	
	`-1:` probing on all `NumBranch` variables.	
	n: probing on n variables.	
PBin	Probing on binary variables option.	0
	`0:` no probing on binary variables.	
	`1:` probing on binary variables.	
PXDo	Number of probing variables fully optimized (not fixed at bound).	−1
PStart	Level of branch-and-reduce tree where probing begins.	0
	`0:` probing begins at root node.	
	n: probing begins at level n.	
PEnd	Level of branch-and-reduce tree where probing ends.	−1
	`-1:` probing never ends.	
	n: probing ends at level n.	
PFreq	Level-frequency of probing applications.	3
	`1:` probing is done at every level of the search tree.	
	n: probing is done every n levels, beginning at level `PStart` and ending at level `PEnd`.	
ProFra	Fraction of probe to bound distance from relaxed solution when forced probing is done.	0.67
TwoWays	Determines wether probing on both bounds is done or not.	1
	`0:` probing to be done by farthest bound	
	`1:` probing to be done at both bounds	
MaxRedPass	Maximum number of times range reduction is performed at a node before a new relaxation is constructed. At any given node, at most `MaxRedPass` calls of the range reduction heuristics will be performed for tightening based on feasibility, marginals, and probing in accordance to the options `TDo`, `MDo`, and `PDo`, respectively.	10
MaxNodePass	Maximum number of passes (relaxation constructions) allowed through a node. If postprocessing improves the node's lower bound in a way that satisfies the absolute or relative tolerances, `RedAbsTol` or `RedRelTol`, respectively, the process of lower bounding followed by postprocessing is repeated up to `MaxNodePass` times.	5
RedRelTol	Relative improvement in the objective to reconstruct the relaxation of the current node.	0.1
RedAbsTol	Absolute improvement in the objective to reconstruct the relaxation of the current node.	0.1

5.5 Branching options

Option	Description	Default
BrVarStra	Branching variable selection strategy.	0
	0: BARON's dynamic strategy	
	1: largest violation	
	2: longest edge	
BrPtStra	Branching point selection strategy.	0
	0: BARON's dynamic strategy	
	1: ω-branching	
	2: bisection-branching	
	3: convex combination of ω and bisection as dictated by ConvexRatio	
ConvexRatio	The branching point under BrPtStra = 3 is set to ConvexRatio $*$ $\omega + (1 - \texttt{ConvexRatio}) * \beta$, where ω and β are the ω- and bisection-branching points.	0.7
ModBrpt	Branch point modification option.	1
	0: BrPtStra-dictated branching point is used without any modifications.	
	1: allows BARON to occasionally modify the BrPtStra-dictated branching point, if deemed necessary.	
NumBranch	Number of variables to be branched on.	0
	-1: consider the model variables as well as variables introduced by BARON's lower bounding procedure.	
	0: consider only the original model variables for branching.	
	n: consider only the first n variables for branching.	
	This option requires knowledge about variable orders and is recommended for *advanced users only*.	
NumStore	Number of variables whose bounds are to be stored at every node of the tree.	0
	0: store NumBranch variables	
	-1: store all variables	
	n: store n variables	
	This option requires knowledge about variable orders and is recommended for *advanced users only*.	

5.6 Heuristic local search options

Option	Description	Default
DoLocal	Local search option for upper bounding.	1
	0: no local search is done during upper bounding	
	1: BARON's dynamic local search decision rule	
	$-n$: local search is done once every n iterations	
MaxHeur	Maximum number of passes allowed for local search heuristic, provided the upper bound improvement during two consecutive passes satisfies either the relative or absolute improvement tolerance (see HRelTol and HAbsTol).	5
HabsTol	Absolute improvement requirement in the objective for continuation of local search heuristic.	0.1
HRelTol	Relative improvement requirement in the objective for continuation of local search heuristic.	0.1
NumLoc	Number of local searches done in NLP preprocessing. The first one begins with the user-specified starting point as long as it is feasible. Subsequent local searches are done from judiciously chosen random starting points. If NumLoc is set to -1, local searches in preprocessing will be done until proof of globality or MaxPreTime is reached. If NumLoc is set to -2, BARON decides the number of local searches in preprocessing based on problem and NLP solver characteristics.	-2
LocRes	Option to control output to log from local search.	0
	0: no local search output.	
	1: detailed results from local search will be printed to res.dat file.	

5.7 Output options

Option	Description	Default
PrFreq	Log output frequency in number of nodes.	1000
PrTimeFreq	Log output frequency in number of seconds.	30
PrLevel	Level of results printed. A larger value produces more output.	1
	≤ 0: all log output is suppressed	
	> 0: print log output	
DotBar	Name of BARON problem file to be written.	
ObjName	Name of objective variable to be optimized. By default, BARON writes the complete objective function to be optimized. If the user specifies an ObjName, this will be written in place of an objective function in the DotBar file, provided a Reform level of 0 is used. Useful only in conjunction with DotBar and if Reform is set to 0.	
Reform	Reformulation level of problem. A value of 0 indicates no reformulation: the complete objective function is listed as an additional constraint and the model minimizes an objective variable. A value of 1 replaces the objective variable by the objective constraint. This is sometimes useful for reducing the model size. A larger Reform value indicates a more aggressive reformulation (if possible).	100

5.8 Other options

Option	Description	Default
eqname.equclass	Specifies nature of constraint named **eqname** in the user's model. Slices like "**supply.equclass('new-york') 1**" are allowed.	0
	0: Regular constraint.	
	1: Relaxation-only constraint. These constraints are provided to BARON as **RELAXATION_ONLY_EQUATIONS** and used to help strengthen the relaxation bound but are not considered as part of the user model and thus not used for feasibility testing of solutions or local search. Adding, for instance, the first-order optimality conditions as relaxation-only constraints often expedites convergence.	
	2: Convex constraint. These constraints are provided to BARON as **CONVEX_EQUATIONS** and used to generate cutting planes from the set of outer approximating supporting hyperplanes of the convex constraint set.	
	3: Convex constraint that is relaxation-only.	
LPSol	Specifies the LP solver to be used.	3
	3: CPLEX	
	7: XPRESS	
	8: CLP	
LPAlg	Specifies the LP algorithm to be used.	0
	0: automatic selection of LP algorithm	
	1: primal simplex	
	2: dual simplex	
	3: barrier	
NLPSol	Specifies the NLP solver to be used.	2
	2: MINOS	
	4: SNOPT	
	6: GAMS NLP solver (see ExtNLPsolver)	
	9: IPOPT	
ExtNLPSolver	Specifies the GAMS NLP solver to be used when NLPSol is set to 6. All GAMS NLP solvers are available through this option. If a non-existing solver is specified or the solver chosen cannot solve NLPs, NLPSol will be reset to its default.	CONOPT
BasKp	Indicates whether basis information is to be saved.	1
	0: no basis information is saved	
	1: LP solver working basis will not be modified if at least **basfra** $* n$ of its basic variables are also basic in the saved basis for the node that is about to be solved.	
BasFra	Similarity measure between bases for basis update not to occur.	0.7
InfBnd	Infinity value to be used for variable bounds. If set to 0, then no bounds are used.	0
NodeSel	Specifies the node selection rule to be used for exploring the search tree.	0
	0: BARON's	
	1: best bound	
	2: LIFO	
	3: minimum infeasibilities	

Option	Description	Default
PostAbsTol	Absolute tolerance for postponing a node. See PostRelTol.	1e30
PostRelTol	Relative tolerance for postponing a node.	1e30
	Instead of branching after solving a node, it is often advantageous to postpone the current node if its lower bound is sufficiently above the (previously) second best lower bound in the branch-and-bound tree. Let z and $z2$ denote the current node's lower bound and the previously second best lower bound in the branch-and-bound tree, respectively. Postponement of a node will take place if any of the following two conditions holds: • $z - z2 \geq$ PostAbsTol • $z - z2 \geq$ PostRelTol $\times |z2|$	
PreLPDo	Number of preprocessing LPs to be solved in preprocessing.	1
	-n: preprocess the first n problem variables 0: no preprocessing LPs should be solved 1: preprocess all problem variables including those introduced by BARON's reformulator 2: preprocess the first NumStore problem variables 3: preprocess all original problem variables	
CutOff	Ignore solutions that are no better than this value. Can also be used as GAMS model suffix option: *(modelname).cutoff = (value)*.	∞