BARON:
Branch and Reduce Optimization Navigator

A High Level Overview for CME334
Assembled by Thomas Lipp
What is BARON?

- BARON is a branch and bound non-linear, mixed integer global optimization solver dating back to 1991, although last updated in 2010
- Developed at University of Illinois at Urbana-Champaign by Nikolaos Sahinidis.
- Problems can be stated in either the AIMMS or GAMS modeling languages, although computing time is available on the NEOS server for GAMS
- Objective and constraint functions must be factorable
- All non-linear variables and expressions must be bounded from above and below
A quick reminder of Branch and Bound algorithms.

From:
What does the Baron Algorithm Look like?

From:
What does each step do?

• Unfortunately, the Baron documentation does not address each step fully. BARON is written to be modular so that user subroutines can be used to replace certain steps as needed.

• The creators are also trying to market BARON: http://www.theoptimizationfirm.com/

• The subproblems are generally fed to existing solvers: SNOPT, MINOS, CPLEX, etc.
Things I won’t be able to answer

• How does BARON handle mixed integers
• What exactly occurs in any given step
• There may be more information available in:
What does BARON have to offer, then?

• BARON calls itself a branch and reduce algorithm rather than a branch and bound algorithm, and it is this “reducing” that will provide the most insight.

• The modular nature of BARON allows it to be tailored to your specific application.
Preprocessing on a Node

From:
Feasibility Based Range Reduction: a “Poor Man’s LP”

- The solid outer box represents the initial domain of the node.
- The solid inner box represents the reduced domain by considering each of the constraints individually. “The Poor Man’s LP”
- The dotted boundary represents solving for the limits on each coordinate with all constraints active.
- The shaded region is the true feasible region.

From: Sahinidis, N. V. and M. Tawarmalani, BARON 9.0.4: Global Optimization of Mixed-Integer Nonlinear Programs, User’s manual, 2010
Lower Bounding

Lower Bounding

This is an overview of the lower bounding scheme used by BARON, but is a little complex to go over. Remember that one requirement of all of our functions was that they be factorable:

**Algorithm Relax f(x)**

If \( f(x) \) is a function of a single variable \( x \in [x^l, x^u] \), then

Construct under- and over-estimators for \( f(x) \) over \([x^l, x^u]\),

else if \( f(x) = g(x)/h(x) \), then

Fractional Relax \((f, g, h)\)

end of if

else if \( f(x) = \prod_{i=1}^{t} f_i(x) \), then

for \( i := 1 \) to \( t \) do

Introduce variable \( y_{f_i} \), such that \( y_{f_i} = \text{Relax} \ f_i(x) \)

end of for

Introduce variable \( y_f \), such that \( y_f = \text{Multilinear Relax} \ \prod_{i=1}^{t} y_{f_i} \)

else if \( f(x) = \sum_{i=1}^{t} f_i(x) \), then

for \( i := 1 \) to \( t \) do

Introduce variable \( y_{f_i} \), such that \( y_{f_i} = \text{Relax} \ f_i(x) \)

end of for

Introduce variable \( y_f \), such that \( y_f = \sum_{i=1}^{t} y_{f_i} \)

else if \( f(x) = g(h(x)) \), then

Introduce variable \( y_h = \text{Relax} \ h(x) \)

Introduce variable \( y_f = \text{Relax} \ g(y_h) \)

end of if

From:

Note: there are other bounding schemes discussed, but this appears to be the one that is used
Lower Bounding Subroutines

**Algorithm Multilinear Relax** $\prod_{i=1}^{t} y_{r_i}$ (Recursive Arithmetic)
for $i := 2$ to $t$ do
    Introduce variable $y_{r_1,\ldots,r_i} = \text{Bilinear Relax} \ y_{r_1,\ldots,r_{i-1},r_i}$. 
end of for

**Algorithm Bilinear Relax** $y_i y_j$ (Convex/Concave Envelope)
- Bilinear Relax $y_i y_j \geq y_i^u y_j + y_i^l y_j - y_i^u y_j$
- Bilinear Relax $y_i y_j \geq y_i^l y_j + y_i^u y_j - y_i^u y_j$
- Bilinear Relax $y_i y_j \leq y_i^u y_j + y_i^l y_j - y_i^u y_j$
- Bilinear Relax $y_i y_j \leq y_i^l y_j + y_i^u y_j - y_i^u y_j$

**Algorithm Fractional Relax** $(f, g, h)$ (Product Disaggregation)
Introduce variable $y_f$
Introduce variable $y_g = \text{Relax} \ g(x)$
if $h(x) = \sum_{i=1}^{t} h_i(x)$, then
    for $i := 1$ to $t$ do
        Introduce variable $y_{f,h_i} = \text{Relax} \ (y_f h_i(x))$
    end of for
    Introduce relation $y_g = \sum_{i=1}^{t} y_{f,h_i}$
else
    Introduce variable $y_h = \text{Relax} \ h(x)$
    Introduce relation $y_g = \text{Bilinear Relax} \ y_f y_h$
end of if

**Algorithm Univivariate Relax** $f(x_j)$ (Recursive Sums and Products)
if $f(x_j) = c x_j^p$ then
    Introduce variable $y_f = \text{Monomial Relax} \ c x_j^p$
else if $f(x_j) = c p^{x_j}$ then
    Introduce variable $y_f = \text{Power Relax} \ c p^{x_j}$
else if $f(x_j) = c \log(x_j)$ then
    Introduce variable $y_f = \text{Logarithmic Relax} \ c \log(x_j)$
else if $f(x_j) = \prod_{i=1}^{t} f_i(x_j)$, then
    for $i := 1$ to $t$ do
        Introduce variable $y_{f_i}$, such that $y_{f_i} = \text{Relax} \ f_i(x)$
    end of for
    Introduce variable $y_f$, such that $y_f = \text{Multilinear Relax} \ \prod_{i=1}^{t} y_{f_i}$
else if $f(x_j) = \sum_{i=1}^{t} f_i(x_j)$, then
    for $i := 1$ to $t$ do
        Introduce variable $y_{f_i}$, such that $y_{f_i} = \text{Relax} \ f_i(x_j)$
    end of for
    Introduce variable $y_f$, such that $y_f = \sum_{i=1}^{t} y_{f_i}$
end of if
Lower Bounding Bilinearities

\( x_i x_j < 0 \)

\( (x_i^U - x_i)(x_j^U - x_j) \geq 0 \)

\( x_i^U x_j^U - x_i^U x_j - x_i^U x_j + x_i x_j > 0 \)

\( x_i x_j > x_i^U x_j + x_i^U x_j - x_i^U x_j \)

\( x_i x_j > x_i^L x_j + x_i^L x_j - x_i^L x_j \)

So create a new variable to replace the bilinearity

\( w_{ij} < 0 \)

\( w_{ij} > x_i^U x_j + x_i^U x_j - x_i^U x_j \)

\( w_{ij} > x_i^L x_j + x_i^L x_j - x_i^L x_j \)

A similar procedure can be performed in the case \( x_i x_j > 0 \) and for upper bounding.
Lower Bounding Linear Fractional Terms

- The same inequalities used in the bilinear case can be rearranged, with some added technicalities on sign to produce the linear fractional inequalities:

\[
\frac{x_i}{x_j} \geq \frac{x_i^U}{x_j} + \frac{x_i}{x_j^L} - \frac{x_i^U}{x_j^L}
\]

\[
\frac{x_i}{x_j} \geq \frac{x_i^L}{x_j} + \frac{x_i}{x_j^U} - \frac{x_i^L}{x_j^U}
\]

\[
\frac{x_i}{x_j} \leq \frac{x_i^U}{x_j} + \frac{x_i}{x_j^L} - \frac{x_i^U}{x_j^L}
\]

\[
\frac{x_i}{x_j} \leq \frac{x_i^L}{x_j} + \frac{x_i}{x_j^U} - \frac{x_i^L}{x_j^U}
\]
Further Lower Bounding

• Once a lower bound is created, a linear approximation of this bound will often be created to further speed computation.

From:
Upper Bounding

Post Processing on a Node

From:
Optimality Based Range Reduction

• We will first define three different problems:

\[(P) : \min f(x) \quad \text{s.t.} \quad g(x) \leq 0 \quad \text{Original problem}\]

\[(R) : \min \tilde{f}(\bar{x}) \quad \text{s.t.} \quad \tilde{g}(\bar{x}) \leq 0 \quad \text{Relaxed problem}\]

\[(R_y) : \varphi(y) = \min f(x) \quad \text{s.t.} \quad g(x) \leq y \quad \text{Perturbed problem}\]
In this example the upper bound range constraint is active on $x$.

$L$: the solution of the relaxed problem $R$

$U$: an upper bound on the original problem $P$

$\Phi$: $\Phi(y)$ unknown solution of $R(y)$ but is $L$ when $y = 0$, as $R(0) = R$.

$K_j^*$: the range reduction if $\Phi(y)$

$K_j$: the range reduction based on the supporting hyperplane of $\Phi(y)$, $z$

$Z$: a supporting hyperplane based on the Lagrange multiplier in the solution of $R$.

Optimality Based Range Reduction: on an inactive constraint

In this example there is no range constraint active.

L: the solution of the relaxed problem R
U: an upper bound on the original problem P
\(z^L\): linear underestimator found by fixing the value of \(x\) to be \(x_j^L\)
\(z^U\): linear underestimator found by fixing the value of \(x\) to \(x_j^U\)
\(\kappa_j^* / \pi_j^*\): the range reduction if the entire value function were known.
\(\kappa_j / \pi_j\): the range reduction based on the underestimators \(z\).
Branching

Branching: Variable Selection

• BARON uses a rectangular subdivision scheme, so only a single variable is chosen for each branching.
• The variable chosen to branch is the variable that contributes the most to the “relaxation gap”.
• This is a non trivial process as many additional variables were introduced in our underestimating step
Branching: Point Selection

- Periodically, branch at the midpoint
- Otherwise, branch at the solution of the lower bounding problem.
- This decision is stored, but not executed until the node is selected again.
Node Selection

From:
Sahinidis, N. V. and M. Tawarmalani, BARON 9.0.4: Global Optimization of Mixed-Integer Nonlinear Programs, User’s manual, 2010
Node Selection

• Default BARON implementation is a composite value based on lower bound, violation (sum of violations of all variables) and order of creation.

• If memory limits are approached BARON switches to FIFO.

• As with all steps, customizable by the user.
For More Information

• Sahinidis, N. V. and M. Tawarmalani, BARON 9.0.4: Global Optimization of Mixed-Integer Nonlinear Programs, User's manual, 2010

• M. Tawarmalani and N. V. Sahinidis, ‘Global optimization of mixed-integer nonlinear programs: A theoretical and computational study, Math Program, DOI 10.1007/s10107-003-0467-6, 2004

• M. Tawarmalani and N. V. Sahinidis, Convexification and global optimization in continuous and mixed-integer nonlinear programming: theory, algorithms, software, and applications, Springer, 2002