Journal of Global Optimization (2005) 32:259–280
DOI 10.1007/s10898-004-2705-8

Accelerating Branch-and-Bound through a Modeling Language Construct for Relaxation-Specific Constraints

NIKOLAOS V. SAHINIDIS[1] and MOHIT TAWARMALANI[2]
[1]*Department of Chemical and Biomolecular Engineering, University of Illinois at Urbana-Champaign, 600 South Mathews Avenue, Urbana, Illinois 61801, USA*
(e-mail: nikos@uiuc.edu)
[2]*Krannert School of Management, Purdue University, West Lafayette, IN 47907-1310, USA*
(e-mail: mtawarma@mgmt.purdue.edu)

(Received 1 April 2004; accepted in revised form 7 April 2004)

Abstract. In the tradition of modeling languages for optimization, a single model is passed to a solver for solution. In this paper, we extend BARON's modeling language in order to facilitate the communication of problem-specific relaxation information from the modeler to the branch-and-bound solver. This effectively results into two models being passed from the modeling language to the solver. Three important application areas are identified and computational experiments are presented. In all cases, nonlinear constraints are provided only to the relaxation constructor in order to strengthen the lower bounding step of the algorithm without complicating the local search process. In the first application area, nonlinear constraints from the reformulation–linearization technique (RLT) are added to strengthen a problem formulation. This approach is illustrated for the pooling problem and computational results show that it results in a scheme that makes global optimization nearly as fast as local optimization for pooling problems from the literature. In the second application area, we communicate with the relaxation constructor the first-order optimality conditions for unconstrained global optimization problems. Computational experiments with polynomial programs demonstrate that this approach leads to a significant reduction of the size of the branch-and-bound search tree. In the third application, problem-specific nonlinear optimality conditions for the satisfiability problem are used to strengthen the lower bounding step and are found to significantly expedite the branch-and-bound algorithm when applied to a nonlinear formulation of this problem.

Key words: BARON, First-order optimality conditions, Mathematical programming modeling languages, Pooling problem, Reformulation-linearization, Satisfiability

1. Introduction

Powerful algebraic modeling languages have been developed over the last two decades for linear and nonlinear mathematical programs (cf. [3, 6]). These languages permit rapid development of new optimization models and algorithms, as well as convenient experimentation with existing numerical optimization solvers. Their key features include support for arithmetic, logical and conditional expressions, automatic differentiation, and seamless interfaces to popular mathematical programming solvers.

Algorithmic needs in global optimization have recently motivated the development of specialized global optimization modeling languages (BARON [8, 9, 34], NUMERICA [36], and NOP-2 [23]) and have played an important role in the development of other, more general, modeling systems (gProms [31] and MINOPT [24]).

The development of the BARON global optimization system began in the early 1990s in an effort to integrate constraint programming and optimization techniques within the branch-and-bound framework for the global optimization of nonconvex nonlinear and mixed-integer nonlinear programs. The approach relies on constraint propagation, interval arithmetic, and duality to draw inferences regarding ranges of integer and continuous variables in an effort to expedite the traditional branch-and-bound algorithm for global optimization problems. Because considerable emphasis is placed on the reduction of variable bounds, the overall methodology is referred to as *branch-and-reduce* [19, 20]. This approach has been enhanced with a modeling language [8, 9, 34], finite branching schemes [1, 25], and a number of convexification techniques [32, 33, 35]. The implementation of this methodology in BARON and its application in a variety of applications, including chemical process design and operation, chip layout and design, design of just-in-time manufacturing systems, optimization under uncertainty, pooling and blending problems, and molecular design were reviewed recently in [34].

Modeling languages for global optimization serve two primary goals. First, they facilitate the automation and systematic testing of global optimization algorithms. Second, they provide a vehicle for testing new modeling language concepts that are currently not supported by general mathematical programming languages because they are highly specialized or in their formative stages.

The purpose of this paper is to present a modeling language construct that can be used to considerably enhance the convexification capabilities of the BARON global optimization system. The work is motivated by the observation that nonlinear problem reformulations can often tighten the relaxation process of a branch-and-bound algorithm while making local search considerably more difficult. It therefore seems natural that a global optimization solver would benefit from user-supplied information about *two* different optimization models: one that provides upper bounds (local search) and another that provides lower bounds (global search). Equipped with such a tool, the user could provide modeling information that would help the global solver improve the efficiency of its global search component without compromising its local search capabilities.

The remainder of this paper is structured as follows. Section 2 reviews the algorithmic framework of branch-and-reduce and Section 3 provides a short description of the BARON modeling language. In Section 4, we describe a new modeling feature in BARON that allows the user to pass relaxation-only equations to the global solver. Sections 5–7 describe computational experience with the application of this construct in three different areas. Finally, conclusions are drawn and possible extensions are suggested in Section 8.

2. The Branch-and-Reduce Algorithmic Framework

2.1. MIXED-INTEGER NONLINEAR PROGRAMMING

We address the problem of finding a globally optimal solution of the following mixed-integer nonlinear program:

$$\begin{array}{lll} (\mathrm{P}) & \min & f(x) \\ & \text{s.t.} & g(x) \leqslant 0, \\ & & x_i \in \mathbb{Z}, \quad i = 1, \ldots, n_{\mathrm{d}}, \\ & & x_i \in \mathbb{R}, \quad i = n_{\mathrm{d}} + 1, \ldots, n, \end{array}$$

where $f : \mathbb{R}^n \mapsto \mathbb{R}$ and $g : \mathbb{R}^n \mapsto \mathbb{R}^m$ are assumed to be factorable functions [16]. To solve P, BARON first reformulates the problem to an almost separable one and preprocesses it to reduce variable ranges. Then, we follow the prototypical branch-and-bound algorithm of Horst and Tuy [12] by specifying a number of branching and bounding operations. Our approach combines elements of the algorithms of Falk and Soland [5] and McCormick [15]. Furthermore, it incorporates interval analysis and constraint programming tools to expedite the search. The main algorithmic steps are detailed next.

2.1. AUTOMATIC PROBLEM REFORMULATION AND RELAXATION

It is well-known that any factorable program can be converted to a separable program through the introduction of additional variables (cf. [14]). In BARON, the mathematical program provided by the user is first symbolically reformulated to an *almost* separable form. In particular, bilinearities of the original model are retained. The details of this symbolic reformulation and subsequent relaxation are provided in [35] and an example can be found in [22]. A similar restructuring of the original program that retains bilinearities as well as higher order multilinearities facilitates the construction of reformation–linearization technique (RLT)-based lower bounds [30].

Assuming that all problem variables are bounded, a lower bounding program can be easily derived from the above reformulation. A univariate concave (convex) function $f(x)$, where $x \in [x^L, x^U]$, is underestimated (overestimated) by the chord that joins the points $(x^L, f(x^L))$ and $(x^U, f(x^U))$. Conversely, a univariate concave (convex) function is overestimated (underestimated) by the function itself. Then, a convex outer-approximator of any concave or convex univariate function can be constructed by combining these estimators. Similar arguments allow construction of outer-approximators of arbitrary univariate lower semi-continuous functions. Bilinear terms (xy) over a rectangle $[x^L, x^U] \times [y^L, y^U]$ are outer-approximated via four hyperplanes due to [15].

The above-described reformulation–relaxation scheme was first proposed in [19, 20] and was inspired from the relaxation technique of McCormick who developed an algebra of outer-estimators for bounding factorable programs [15]. McCormick's scheme produces the same outer-approximator as the above relaxation technique when applied to the almost separable reformulation of a mathematical program with no common subexpressions. However, McCormick's relaxation differs from the above as it is expressed using *nondifferentiable* operators and does not introduce any intermediate variables for subexpressions. It was proven recently in [35] that our reformulation–relaxation scheme dominates the McCormick scheme by enabling the identification of common subexpressions and, as a result, provides tighter relaxations when applied to the complete mathematical program P.

After its original proposal in [19, 20], the reformulation–relaxation described above has been adopted by several branch-and-bound systems. For instance, it is used in gProms [31] and LINGO [7]. While this type of relaxation is still available as an option in BARON, BARON's current default relaxation strategy is to *further relax* the convex nonlinear functions of the above-described relaxation by tangential outer-approximators based on a variant of the sandwich algorithm as proposed in [35]. This yields an *entirely polyhedral outer-approximator* that can be solved efficiently and reliably via linear programming techniques.

2.3. ROOT NODE PREPROCESSING

Construction of the relaxation described in the previous subsection requires a bounded box over the problem variables. To obtain such bounds, each problem variable is minimized and then maximized over the linear constraint set of the reformulation described in Section 2.2. In addition, as proposed in [19, 20], the nonlinear constraints from the reformulation are used to propagate variable bounds using interval arithmetic. This constraint propagation step produces tighter variable bounds than obtained by just solving linear programs (LPs).

Local searches are performed from several randomly generated starting points. The starting points for these local searches are generated in the bounding box obtained as a result of the preprocessing on variable bounds. Subsequently, an objective function cut of the form $f(x) \leqslant f^*$ is used to further restrict variable bounds in the constraint propagation phase. Here, f^* corresponds to the value of the incumbent and constraint propagation is done whenever a new upper bound is found. Furthermore, whenever a new upper bound is found and/or constraint propagation improves lower/upper bounds for some of the problem variables, the above-mentioned preprocessing LPs are solved approximately by doing Fourier–Motzkin elimination using one constraint at a time as detailed in [25]. Empirical investigation in [25] demonstrated that this approximate method of solving the preprocessing LPs works well when initial bounds on the problem variables are obtained by solving the most of LPs exactly.

2.4. NODE POSTPROCESSING

Once the relaxation is solved, the optimal Lagrange multipliers provide the slope of a first-order underestimator of the relaxation value function in the vicinity of the relaxation solution. As proposed in [19, 20], this underestimator is used to restrict ranges of variables. Subsequently, LPs are solved to obtain tighter bounds on a selected set of variables in a step termed as "probing" in BARON. Probing LPs differ from the preprocessing LPs in that they include polyhedral outer-approximators of the nonlinear relations in the reformulation. Two types of LPs are considered in probing. Normally, a variable will be minimized and maximized over the linear constraints. Optionally, a variable is fixed at some point between the bounds, an LP is solved to optimize the linear underestimator of the objective, and Lagrange multipliers are used to tighten the range for the variable. Probing LPs are solved only for variables that contribute significantly to the gap between the original nonconvex problem functions and their underestimators at the relaxation solution. For the remaining problem variables, three methods of tightening variable bounds are used. First, as shown in [35], dual solutions from the probing LPs provide first-order underestimators for the value function of the original problem and are used to facilitate range contraction. Second, probing LPs are solved approximately using Fourier–Motzkin elimination as mentioned above on the linear part of the reformulation to propagate the improved bounds. Third, constraint propagation is done on the nonlinear constraints.

2.5. NODE PARTITIONING

Rectangular partition is used to divide the range of one variable into two subsets. Using certain heuristic strategies proposed in [35], the underestimation gap – between the nonconvex functions of the reformulation and the convex relaxation – at the optimal solution of the relaxation is apportioned among the problem variables. The branching variable is then selected based on its contribution to the underestimation gap while taking into account its interval length. Although branching in BARON can optionally occur in the space of the introduced variables, BARON, by default, will branch only on the original problem variables. A violation transfer scheme proposed in [35] is used to transfer the contributions of introduced problem variables to original problem variables. The violation transfer scheme takes into account the relative importance of the constraints by suitably weighting their underestimating gaps by constraint duals.

Once a variable has been selected for branching, the branching point is determined as a convex combination of the relaxation solution and the mid point of the variable's range. This guarantees exhaustiveness, which is required for proving convergence of branch-and-bound [12]. Yet, our approach biases the branching point selection towards the relaxation solution, which often quickly reduces the relaxation gap.

Finally, if the best known solution lies in the interior of the current box, this solution point is chosen for branching. For certain branch-and-bound processes, this modifying branching point selection rule guarantees finiteness as opposed to convergence at the limit only [25].

2.6. NODE SELECTION

A node corresponding to the least lower bound is chosen, at least occasionally, for further processing and partitioning. This is needed in order to guarantee convergence when branching occurs in continuous variable spaces [12]. Occasionally, BARON will instead select for further processing a node for which the violation – between underestimators and original nonlinear problem functions – at its relaxation solution is small. This often leads to the selection of nodes whose relaxation identifies feasible solutions of P early during the search. Finally, if memory requirements dictate it, BARON will switch to a last-in, first-out (LIFO) strategy.

2.7. NODE PREPROCESSING

During the course of the branch-and-bound search, each node is preprocessed in a manner that resembles the root node preprocessing described in Section 2.3. However, the preprocessing LPs are not solved exactly and the local search is done only sporadically when deemed potentially beneficial. The approximate solution of LPs and constraint propagation of the bounds using interval arithmetic techniques are inexpensive computationally. Nonetheless, they greatly reduce variable ranges and are therefore performed at every node of the search tree.

2.8. IMPLEMENTATION

Initially, the branch-and-bound algorithm was encoded in the `BARON` system and specialized modules were developed for solving various special classes of global optimization problems [19–21]. Later, the modeling language was developed [8, 9] along with preliminary capabilities of reformulation and relaxation of factorable programs. More recently, the solver and modeling language underwent considerable rewrite to include sophisticated data structures and several algorithmic enhancements, including convexification and range reduction strategies [34, 35].

At present, `BARON`'s core consists of nine major components: (1) the I/O handler, which is described in more detail in the next section; (2) the preprocessor, which aims to obtain a bounded box for the problem variables and a good feasible solution for the problem; (3) the navigator, which coordinates the transitions between the different states of branch-and-bound; (4) the data organizer; (5) range reduction facilities; (6) Interfaces to `OSL`, `CPLEX`, `MINOS`, `SNOPT`, and `SDPA` for solving local and underestimating problems; (7) `BARON`'s automatic differentiator; (8) sparse matrix utilities; and (9) debugging and development facilities.

3. BARON's Modeling Language

A context-free grammar has been developed as part of `BARON` for the input of factorable nonlinear programming problems in which the constraint and objective functions are recursive sums and products of univariate functions of monomials, powers, and logarithms. A `BARON` input file has the following components:

- An `OPTIONS` section, which is optional and allows the user to select values for options that control termination, range reduction, branching, and numerous other algorithmic steps.
- An optional command to specify the maximum amount of memory that may be used by `BARON`'s data structures.
- A `MODULE` command that allows the user to select from amongst `BARON`'s specialized modules or the factorable nonlinear programming solver, which is denoted as `NLP`.
- The problem data section, where the user can use commands to:
  - declare `VARIABLES`, `BINARY_VARIABLES`, `INTEGER_VARIABLES`;
  - optionally provide `LOWER_BOUNDS` and `UPPER_BOUNDS` for some or all problem variables;
  - optionally specify `BRANCHING_PRIORITIES` for problem variables;
  - declare `EQUATIONS` of the mathematical program to be solved;
  - describe declared `EQUATIONS`;

- describe an objective function and specify whether the objective is to be minimized or maximized;
- optionally provide a STARTING_POINT.

As an example, consider the following problem from [17]:

$$\begin{aligned} \min \quad & -500x + 2.5x^2 + \cdots + 0.8x^{50} \quad (\text{see}[17]) \\ \text{s.t.} \quad & 1 \leqslant x \leqslant 2. \end{aligned} \tag{1}$$

The BARON input for this problem is then as follows:

```
MODULE: NLP;
VARIABLE x;
LOWER_BOUND { x: 1; }
UPPER_BOUND { x: 2; }
OBJ: minimize − 500 * x + 2.5 * x^2 + ··· + 0.8*x^50;
```

4. New Modeling Construct for Problem-Specific Lower Bounding

The relaxation constructor described in Section 2 does not make use of any optimality conditions. One observes, though, that the example of the previous section will have a solution either at one of the two bounds of x or at a point where the gradient of the objective is zero, *i.e.*, a point satisfying:

$$0 = 500 + 5x + \cdots + 40x^{49}. \tag{2}$$

One can, therefore, solve the above example problem by first evaluating the objective function at the two bounds of x and then by optimizing the polynomial subject to constraint (2).

Without constraint (2), BARON finds the global minimum of (1) at the root node and takes 29 nodes to prove globality. When the optimality condition (2) is added as a constraint, BARON benefits considerably as it terminates with proof of globality at the root node. On the contrary, the effect of constraint (2) to the local solver GAMS/MINOS is detrimental. In particular, when the problem is solved without constraint (2), GAMS/MINOS finds the global solution in five iterations. On the other hand, when the local optimality condition (2) is added as a constraint, the local solver takes 15 iterations and declares the problem as locally infeasible. Both GAMS/MINOS runs used the same starting point $x = 1$, which is very close to the global solution of $x = 1.09117$.

Note that BARON also makes use of MINOS in its local search component. Thus, apparently, addition of the optimality condition affects the global search component of the algorithm. In particular, feasibility-based range reduction in conjunction with the objective function cut is able to prove globality within BARON's default 10^{-6} optimality tolerance.

The conclusion drawn from this example is that the addition of constraint (2) greatly accelerates the global search component of BARON while it severely interferes with local search. Extensive computational experiments in the next three sections present further empirical evidence under many different scenarios that corroborate these findings. This leads us to believe that it is beneficial to have the ability of communicating two different models to the global solver: one model to be used for relaxation and range reduction and another model to be used for local search. This can be achieved easily by extending BARON's modeling language to accept the following construct:

```
RELAXATION_ONLY_EQUATIONS < constraint names >;
```

Constraints that are declared with the above construct are processed through BARON's reformulator and relaxation constructor. They are then used for constraint propagation. However, they are not made part of the local search model.

A similar construct can be introduced to achieve a complete separation between the relaxation and local search models:

```
LOCAL_ONLY_EQUATIONS < constraint names >;
```

Here, we restrict discussion to the RELXATION_ONLY_EQUATIONS construct, which can be used to model the above polynomial example as follows:

```
MODULE: NLP;
VARIABLE x;
LOWER_BOUND{ x: 1; }
UPPER_BOUND{ x: 2; }

RELXATION_ONLY_EQUATION e1;
e1 : 500 == 5 * x + ··· + 40*x^49;

OBJ : minimize − 500 * x + 2.5 * x^2 + ··· + 0.8*x^50;
```

The next three sections provide additional examples illustrating the use of this modeling construct and demonstrating its usefulness.

5. Linearizing RLT Constraints and Application to the Pooling Problem

The RLT [26, 27] is a systematic methodology for deriving tight formulations for combinatorial and continuous global optimization problems. In the reformulation step of RLT, constraints of the initial formulation are multiplied with each other to yield valid nonlinear constraints. In the linearization step of RLT, new variables are introduced to replace products of variables. Additional RLT passes may be performed on the newly derived formulation to further strengthen its relaxation.

It has been shown that RLT leads to tight linear programming relaxations, including convex hulls for certain problem classes. Relaxation tightness comes at the expense of a significantly increased problem size that results from the reformulation and linearization steps. Thus, it would be highly desirable to develop an automated procedure for implementing and testing different RLT schemes. In many problem instances, only a few of the RLT constraints may suffice to tighten the formulation and the modeler may thus be interested in studying the effect of a few constraint products on relaxation tightness. Even then, there are still many constraints to be introduced in the linearization phase. As `BARON`'s reformulator currently provides an automated means for the linearization phase, one possibility is to supply `BARON` with the nonlinear formulations that result from the reformulation phase of RLT. A potential complication is that these nonlinear constraints may make local search difficult. Thus, the `RELAXATION_ONLY_EQUATIONS` construct seems ideal for experimenting with different RLT constructs. This is demonstrated in this section using the pooling problem as an example.

The pooling problem is a network flow problem over three sets of nodes: supply nodes that represent the sources of raw materials that flow to demand nodes (final product destinations) either directly or indirectly through transshipment nodes (pools). The unit costs as well as attributes, such as component concentrations of raw materials and final products are given. The problem is then to find the optimal flows in the network so as to maximize net profit. Nonlinearities arise in attribute balances around pools since the pool attribute qualities as well as the inflows and outflows are all variables.

To describe models for the pooling problem, we will use the notation of Table 1.

The so-called q-formulation of the pooling problem is as follows [2]:

$$\begin{aligned}
\min \quad & \sum_{j=1}^{J}\left(\sum_{l=1}^{L} y_{lj}\sum_{i=1}^{I} c_i q_{il} - d_j\sum_{l=1}^{L} y_{lj} + \sum_{i=1}^{I} c_i z_{ij} - \sum_{i=1}^{I} d_j z_{ij}\right)\\
\text{s.t.} \quad & \sum_{l=1}^{L}\sum_{j=1}^{J} q_{il} y_{lj} + \sum_{j=1}^{J} z_{ij} \leqslant A_i \quad i = 1,\ldots,I,\\
& \sum_{j=1}^{J} y_{lj} \leqslant S_l \quad l = 1,\ldots,L,\\
& \sum_{l=1}^{L} y_{lj} + \sum_{i=1}^{I} z_{ij} \leqslant D_j \quad j = 1,\ldots,J,\\
& \sum_{l=1}^{L}\left(\sum_{i=1}^{I} C_{ik} q_{il} - P^{\mathrm{U}}_{jk}\right) y_{lj} + \sum_{i=1}^{I}(C_{ik} - P^{\mathrm{U}}_{jk}) z_{ij} \leqslant 0 \quad k = 1,\ldots,K;\\
& \qquad\qquad j = 1,\ldots,J,
\end{aligned}$$

Table 1. Pooling problem nomenclature

Indices	
i	Raw materials, $i = 1, \ldots, I$
j	Products, $j = 1, \ldots, J$
k	Qualities, $k = 1, \ldots, K$
l	Pools, $l = 1, \ldots, L$
Variables	
q_{il}	Quality of ith product in pool l ($x_{il} = q_{il} \sum_{j=1}^{J} y_{lj}$)
x_{il}	Flow of ith raw material into pool l
y_{lj}	Total flow from pool l to product j
z_{ij}	Direct flow of raw material i to product j
Parameters	
c_i	Unit cost of the ith raw material
d_j	Price of jth product
A_i	Availability of ith raw material
C_{ik}	kth quality of raw material i
D_j	Demand of jth product
P_{jk}^{U}	Upper bound on kth quality of jth product
S_l	lth pool capacity

$$\sum_{i=1}^{I} q_{il} = 1 \quad l = 1, \cdots, L,$$
$$q_{il} \geqslant 0, y_{lj} \geqslant 0, z_{ij} \geqslant 0 \qquad \forall (i, j, l).$$

We now consider the following two sets of constraints:

$$\sum_{i=1}^{I} q_{il} y_{lj} = y_{lj} \qquad l = 1, \ldots, L; \quad j = 1, \ldots, J, \tag{3}$$

$$\sum_{j=1}^{J} q_{il} y_{lj} \leqslant q_{il} S_l \quad i = 1, \ldots, I; \quad l = 1, \ldots, L. \tag{4}$$

These constraints can be obtained when the RLT is applied to the above pooling formulation (see [18] and Chapter 10 of [34]). These two particular constraint sets form a small subset from amongst all the possible RLT constraints that can be generated for the pooling problem. In [34], it was shown that these two sets of constraints play a special role in terms of producing a tight relaxation for the pooling problem by convexifying multilinear expressions of q and y over the constraints $\sum_{i=1}^{I} q_{il} = 1,\ l = 1, \ldots, L$. It is therefore desirable to add (3) and (4) to the pooling problem formulation. When these constraints are added to the q-formulation, we will refer to the resulting model as the pq-formulation.

Tables 2 and 3 present computational results for pooling problems from the literature using local search solvers. The original sources of the problems can be found in Chapter 10 of [34]. In all cases, the problems are solved under GAMS using default starting points and algorithmic options for the solvers. Constraints (3) and (4) are redundant as far as the pooling problem for-

Table 2. Computational results for pooling problems with GAMS/CONCOPT

Problem	*q*-Formulation			*pq*-Formulation		
	Objective	CPUs	Iter	Objective	CPUs	Iter
adhya1	−68.74	0.01	9	−56.67	0.00	5
adhya2	0.00	0.01	4	0.00	0.01	3
adhya3	−65.00	0.03	12	−57.74	0.02	7
adhya4	−470.83	0.01	9	−470.83	0.02	9
bental4	0.00	0.01	3	0.00	0.00	3
bental5	−2900.00	0.02	9	−2700.00	0.03	18
foulds2	−1000.00	0.00	6	−600.00	0.01	14
foulds3	−6.50	0.04	6	−6.50	0.09	9
foulds4	−6.00	0.04	6	−6.50	0.16	23
foulds5	−7.00	0.04	7	−6.50	0.06	7
haverly1	−400.00	0.00	5	0.00	0.00	3
haverly2	−400.00	0.00	5	0.00	0.01	3
haverly3	−750.00	0.01	8	0.00	0.00	3
rt97	Inf	0.00	4	−4330.78	0.01	8
Sum	−6074.07[a]	0.22	89	−3904.74[a]	0.41	107

[a]Not including rt97.

mulation is concerned. Nonetheless, their addition to the problem formulation deteriorates performance of GAMS/CONOPT as shown in Table 2. In particular, this solver finds worse local optima for 8 of these 14 pooling problems while its CPU time almost doubles. Only in one instance (rt97), an improved solution is found by GAMS/CONOPT with the *pq*-formulation. As shown in Table 3, the performance of GAMS/MINOS seems to be somewhat less sensitive to the addition of constraints (3) and (4). In particular, the objective

Table 3. Computational results for pooling problems with GAMS/MINOS

Problem	*q*-Formulation			*pq*-Formulation		
	Objective	CPUs	Iter	Objective	CPUs	Iter
adhya1	0.00	0.01	9	0.00	0.01	9
adhya2	0.00	0	9	0.00	0.01	9
adhya3	0.00	0.01	15	0.00	0.01	15
adhya4	−470.83	0.04	74	−470.83	0.03	59
bental4	−100.00	0	5	−100.00	0.01	5
bental5	−1900.00	0	11	−1900.00	0.02	11
foulds2	−700.00	0.01	15	−700.00	0.01	15
foulds3	−7.50	0.67	420	−8.00	1.01	438
foulds4	−8.00	0.66	463	−8.00	0.83	462
foulds5	−8.00	0.45	305	−8.00	0.49	219
haverly1	−100.00	0.01	5	−100.00	0.01	5
haverly2	−600.00	0.01	5	−600.00	0	5
haverly3	−125.00	0	13	−125.00	0.01	13
rt97	−4391.83	0.02	103	−4391.83	0.02	111
Sum	−8411.1633	1.89	1452	−8411.6633	2.47	1376

Table 4. Computational results for pooling problem with BARON

Problem	Strategy 1				Strategy 2				Strategy 3			
	N_t	N_o	N_m	CPUs	N_t	N_o	N_m	CPUs	N_t	N_o	N_m	CPUs
adhya1	573	550	50	17	30	24	7	1	28	24	7	0.5
adhya2	501	338	41	20	17	13	4	1	17	13	4	0.5
adhya3	9248[a]	2404	1800[a]	1200[a]	31	1	6	1.5	31	1	6	1.5
adhya4	6129[a]	−1	1620[a]	1200[a]	1	1	1	1.5	1	1	1	1
bental4	101	101	14	0.5	1	−1	1	0.5	1	−1	1	0.5
bental5	6445[a]	901	3815[a]	1200[a]	−1	−1	0	0.5	−1	−1	0	0
foulds2	1061	977	106	16	−1	−1	0	0	−1	−1	0	0
foulds3	348[a]	91	260[a]	1200[a]	−1	−1	0	1	−1	−1	0	5
foulds4	326[a]	262	246[a]	1200[a]	−1	−1	0	1	−1	−1	0	1
foulds5	389[a]	316	287[a]	1200[a]	−1	−1	0	1	−1	−1	0	1
haverly1	25	6	5	0	1	1	1	0	1	1	1	0
haverly2	17	1	5	0	1	1	1	0	1	1	1	0
haverly3	3	1	2	0	1	1	1	0	1	1	1	0
rt97	5629	2836	609	173.5	13	6	4	0.5	13	6	4	0.5
Sum	30795	8783	8860	7427	91	42	10	10	89	42	26	12

[a]Run did not terminate within 1200 s.

function quality improves in one case, while the CPU time increases by 30% over the entire collection after the introduction of these constraints.

Computational results with the two formulations and BARON are shown in Table 4. Three strategies are compared. Strategies 1 and 2 correspond to solving the q- and pq-formulations, respectively. Strategy 3 solves the pq-formulation where constraints (3) and (4) are introduced through the RELAXATION_ONLY_EQUATIONS construct. For each strategy, this table shows:

- N_t: The total number of nodes in the search tree. An entry of −1(0) indicates that the problem was solved at the root node after preprocessing (postprocessing).
- N_o: The node where the optimal solution was found.
- N_m: The maximum number of nodes that had to be stored in memory.
- CPUs: CPU seconds on a Dell Precision 530 workstation with a 1.7 GHz Pentium IV Xeon processor. All runs were restricted to 32 MB of memory.

As seen from Table 4, solving the q-formulation results in very slow convergence. The runs for adhya3, adhya4, bental5, foulds3, foulds4, and foulds5 were terminated after 20 min with a large relaxation gap. On the other hand, the pq-formulation makes all these problems easily solvable. With the exception of adhya1, adhya2, adhya3, and rt97, all other problems are solved at the root node. For the problems solved at the root node, using the RELAXATION_ONLY_EQUATIONS construct for the RLT constraints generally reduces the CPU time by reducing the CPU time for local search. The only exception is foulds3, which was solved in the first local search in Strategy 2 whereas additional local searches were required in preprocessing of Strategy 3. Note that `BARON` makes use of `MINOS` for local search and, as seen in Table 3, foulds3 is the only problem in this collection for which `MINOS` benefits from the introduced constraints. Without these constraints, some additional local searches are required to identify the global solution at the root node by Strategy 3, thus increasing the CPU time in comparison to Strategy 2. Nonetheless, for adhya1, adhya2, adhya4, and bental5, which are not solvable at the root node by any strategy, we observe that Strategy 3 is able to terminate faster than Strategy 2. This is because nonlinear RLT constraints are not present in local search in Strategy 3, thus making local search easier for this strategy.

It is interesting to note that the CPU times for solving this set of pooling problems to global optimality (Strategies 2 and 3 in Table 4) are no more than five times as much as the CPU times for local search (Table 3), indicating that these strategies make global search almost as fast as local search for this set of pooling problems.

6. First-Order Optimality Conditions and Application to Unconstrained Polynomial Programs

The example of Section 3 illustrated that adding the first-order optimality conditions to the model formulation can significantly enhance `BARON`'s global search component. In general, it would be interesting to study the effect of adding $\nabla(f(x)) = 0$ as a constraint to the problem of globally minimizing $f(x)$. While optimality constraints have long been used for exclusion tests in interval global solvers such as `GLOBSOL` [13] and `NUMERICA` [36], they have not heretofore been explored for relaxation construction or tightening. In this section, we present the first such computational experimentation. We do so in the context of univariate polynomial problems:

$$\text{(P1)} \qquad \min \ f(x) = \sum_{i=0}^{n} c_i x^i \qquad \text{s.t.} \quad x^{\mathrm{L}} \leqslant x \leqslant x^{\mathrm{U}},$$

where c_i, $i = 1, \ldots, n$ are given real numbers. As one can evaluate the objective at x^{L} and x^{U} and compare their values to the solution of an algorithm that solves the problem in $(x^{\mathrm{L}}, x^{\mathrm{U}})$, without loss of generality, we may restrict attention to $(x^{\mathrm{L}}, x^{\mathrm{U}})$ and reformulate the problem as:

$$\begin{aligned} \text{(P2)} \qquad \min \quad & f(x) = \sum_{i=0}^{n} c_i x^i \\ \text{s.t.} \quad & \sum_{i=1}^{n} i c_i x^{i-1} = 0, \\ & x^{\mathrm{L}} \leqslant x \leqslant x^{\mathrm{U}}, \end{aligned}$$

Table 5. Local search for univariate polynomial programs

Problem	n	f *	GAMS/CONOPT		GAMS/MINOS	
			P1	P2	P1	P2
p1	6	−29763.23	0.10	−29763.23	0.10	−29763.23
p2	50	−663.50	−663.50	Infeasible	−663.50	Infeasible
p3	5	−443.67	0.00	Infeasible	0.00	Infeasible
p4	4	0.00	0.00	0.00	0.00	0.00
p5	4	0.00	0.00	0.30	0.00	0.30
p6	4	7.00	250.00	7.00	250.00	7.00
p7	4	−7.50	0.00	6.00	0.00	6.00

Compared to P1, P2 has an additional constraint that is nothing else but the first-order necessary optimality condition for the unconstrained univariate polynomial program.

Table 5 shows the effect that adding the local optimality condition has on local search. The original sources of the seven problems considered here can be found in [20]. The table shows the order of the polynomial (n), the objective function value of the global minimum (f^*), and the values of the local solutions obtained by GAMS/CONOPT and GAMS/MINOS for models P1 and P2. When the first-order optimality conditions are added to the model formulation, better solutions are found for problems p1 and p6. On the other hand, the solutions for problems p5 and p7 worsen. Worse though, model P2 is declared as locally infeasible for problems p2 and p3 by both local solvers. Apparently, the nonconvexity of the first-order optimality condition is responsible for this bad behavior of the local solvers.

Computational results with the two formulations and BARON are shown in Table 6. Three strategies are compared. Strategies 1 and 2 solve formulations P1 and P2, respectively. Strategy 3 solves formulation P2 where the first-order optimality constraint is introduced through the RELAXATION_ONLY_EQUATIDN construct. As in the previous section, for each strategy, Table 6 shows the total number of nodes (N_t), the node where the

Table 6. Global search with BARON on unconstrained polynomial programs

Problem	Strategy 1			Strategy 2			Strategy 3		
	N_t	N_o	N_m	N_t	N_o	N_m	N_t	N_o	N_m
p1	51	−1	8	1	−1	1	1	−1	1
p2	29	1	5	3	−1	2	3	1	2
p3	45	10	7	1	1	1	1	−1	1
p4	51	−2	6	7	−1	2	7	−1	2
p5	15	−2	4	9	−1	3	9	−1	3
p6	95	−1	11	7	4	3	7	−1	3
p7	19	19	5	1	−1	1	1	−1	1
Sum	305	24	46	29	0	13	29	−5	13

optimal solution is found (N_o), and the maximum number of nodes in memory during the search (N_m). The CPU times were negligible for all these runs.

As seen in Table 6, BARON's global search benefits considerably for all these problems from the addition of the first-order optimality conditions. In particular, Strategies 2 and 3 require 90% fewer total nodes than Strategy 1. In addition, BARON's local search component benefits from the RELAXATION_ONLY_EQUATION construct in the context of problem p6, for which the best solution is now identified earlier during the search.

7. Problem-Specific Optimality Conditions and Application to the Satisfiability Problem

The two previous sections considered generally applicable strategies for generating valid constraints for a given optimization problem. In many problems, the modeler can construct problem-specific constraints which when added to a standard formulation do not alter the solution. For instance, many scheduling and facility location problems possess symmetric solutions that can be eliminated through the addition of appropriate constraints to the problem formulation (cf. [28]). Such constraints can significantly accelerate exact branch-and-bound algorithms. On the other hand, the addition of such constraints may make local search a lot more difficult. In such cases, the RELAXATION_ONLY_EQUATIONS construct becomes useful. We demonstrate this in the context of the famous satisfiability problem.

The satisfiability problem involves a set of boolean variables (x_i, $i = 1,\ldots, n$) and a set of clauses (C_j, $j = 1,\ldots, m$). Each clause consists of literals (variables or their negations) combined by logical *or* connectives. The goal of the satisfiability problem is to determine whether there exists an assignment of values to variables that makes the following conjunctive normal form (CNF) satisfiable:

$$C_1 \wedge C_2 \wedge \cdots \wedge C_m,$$

where $\wedge$ is the logical *and* connective.

The satisfiability problem is central in the theory of computation. It is a core NP-complete problem that is fundamental in applications in automated reasoning, computer-aided design, machine vision, scheduling, and many other areas (cf. [4]). Many specialized algorithms have been developed for the satisfiability problem. The objective of this section is not necessarily to provide a competitive algorithm but merely to use this famous problem in order to demonstrate the potential benefits of the RELAXATION_ONLY_EQUATIONS construct. For this, we consider the following formulation of the problem:

$$\text{(S1)}\quad \min\ f(x) = \sum_{j=1}^{m}\prod_{i=1}^{n}\alpha_i(x_i)$$
$$\text{s.t.}\quad x_i = 0,\ \text{or } 1,\quad i = 1,\ldots,n,$$

where $\alpha_i(x_i)$ is defined as:

$$\alpha_i(x_i) = \begin{cases} 1 - x_i, & \text{if } x_i \text{ is in clause } C_j,\\ x_i, & \text{if the negative of } x_i \text{ is in clause } C_j,\\ 1, & \text{otherwise.}\end{cases}$$

It is known that a CNF is satisfiable if and only if the globally optimal solution of S1 is zero (cf. [10]). Clearly, then, one can reformulate the problem by requiring each of the summands in the objective function of S1 to be zero:

$$\text{(S2)}\quad \min\ f(x) = \sum_{j=1}^{m}\prod_{i=1}^{n}\alpha_i(x_i)$$
$$\text{s.t.}\quad \prod_{i=1}^{n}\alpha_i(x_i) = 0,\quad j = 1,\ldots,m,$$
$$x_i = 0 \text{ or } 1,\quad i = 1,\ldots,n.$$

As S1 is unconstrained, it is particularly easy to do local search with this model. For instance, one can generate a random starting point $x \in [0, 1]^n$, locally improve this point using *continuous* optimization techniques, and, finally, round the solution point. On the other hand, the presence of the nonlinear constraints in S2 makes local minimization very difficult for this problem despite the fact that these constraints may help local search escape from local minima.

Computational results with the two formulations and BARON are shown in Table 7. The problems solved originate from the DIMACS benchmark set for satisfiability and were taken from SATLIB [11]. All runs were restricted to 32 MB of memory on a Dell Precision 530 workstation with a 1.7 GHz Pentium IV Xeon processor. Three strategies are compared. Strategies 1 and 2 solve formulations S1 and S2, respectively. Strategy 3 solves formulation S2 where the constraints are introduced through the RELAXATION_ONLY_EQUATIONS construct. As in previous sections, for each strategy, this table shows the number of total nodes in the search tree (N_t), node where the optimal solution is found (N_o), maximum number of nodes in memory (N_m). In addition, the table presents total CPU seconds (T_t), CPU seconds spent on preprocessing (T_p), most of which is spent doing local search, and the CPU seconds spent on local search after preprocessing (T_l).

As seen in Table 7, the introduction of the nonlinear constraints in Strategies 2 and 3 reduces the number of nodes (N_t) by 60% and the memory

Table 7. Computational results for satisfiability problems with `BARON`

Problem	Strategy 1						Strategy 2						Strategy 3					
	N_t	N_o	N_m	T_t	T_p	T_l	N_t	N_o	N_m	T_t	T_p	T_l	N_t	N_o	N_m	T_t	T_p	T_l
hole6	6541	472	816	116	0	3	2677	−3	28	43	1	12	2677	−1	28	43	0	2
ii8al	1	1	1	0	0	0	−1	−1	0	1	1	0	4	4	4	1	0	0
ii8a2	24	24	16	80	1	1	−1	−1	0	12	4	1	11	11	9	36	0	0
ii8a3	11	11	10	105	1	1	10	10	10	147	26	5	10	10	10	126	1	1
ii8bl	−1	−1	0	5	1	0	−1	−1	0	71	3	1	−1	−1	0	68	1	1
par8-1-c	8	8	5	11	0	0	8	8	3	7	2	0	8	8	3	5	0	0
par8-2-c	7	7	5	7	0	0	11	11	4	9	2	1	11	11	4	7	0	0
par8-3-c	59	59	34	111	0	0	12	12	4	11	3	0	12	12	4	8	0	0
par8-4-c	123	123	67	156	0	0	21	21	3	15	2	1	21	21	3	14	0	0
par8-5-c	105	105	53	215	0	0	67	67	4	29	3	1	67	67	4	27	1	0
Sum	6878	809	1007	803	2	4	2803	123	56	343	44	20	2820	142	69	334	2	3

requirements (N_m) by 94%. Simultaneously, there is over 57% reduction in the total CPU time requirements (T_t). The benefits of Strategy 3 in comparison to Strategy 2 at a first glance may seem insignificant (less than 1% increase in total number of iterations, accompanied by approximately 3% decrease in total CPU time). A closer look reveals that Strategy 3 virtually eliminates all time spent on local search in preprocessing (T_p) and during the tree (T_i). The net CPU time gain by Strategy 3 versus Strategy 2 is only 3% solely due to problem ii8a2, which appears to be a pathological case for which Strategy 3 takes a while to identify the global minimum. In general, though, Strategy 3 seems to serve its purpose, which is to significantly reduce the time spent on local search without compromising the efficiency of the global search component (N_t and N_m).

8. Conclusions and Extensions

This paper has demonstrated that there are several types of typically nonlinear constraints which, while redundant from the modeling point of view and detrimental to local search solvers, can significantly accelerate a branch-and-bound global optimization algorithm by enhancing its lower bounding capabilities. To allow for proper modeling of such constraints, the RELAXATION_ONLY_EQUATIONS construct was introduced in the modeling language of the BARON system.

The use of the RELAXATION_ONLY_EQUATIONS construct was demonstrated in this paper in the context of RLT constraints, first-order optimality conditions, and problem-specific optimality conditions. While CPU time gains by using this construct as opposed to merely adding the nonlinear constraints to the nonlinear formulation were not significant as far as global search is concerned, it is anticipated that the mere presence of such a modeling language construct will motivate modelers to conceive additional applications of this construct.

For RLT, the new modeling construct removes a considerable obstacle towards systematic experimentation with it. The modeler now needs to provide only the nonlinear constraints from the initial step(s) of RLT; BARON will then automatically construct the linearized RLT constraints while avoiding detrimental effects of the nonlinear constraints on local search. A natural extension in this context would be to automate the reformulation step of RLT through a suitable modeling language construct, and combine this with RLT-based relaxations of first-order optimality conditions, and constraint-filtering mechanisms such as those proposed in [29].

Several additional extensions of this work are possible. For instance, the addition of first-order necessary optimality conditions as part of the model constraints was demonstrated to yield significant benefits. Thus, the automatic generation of relaxations of optimality conditions for any user-provided model is worth future investigation. Another possibility is to convexify the constraints passed as RELAXATION_ONLY_EQUATIONS and only then use them as part of the local search model. We conjecture that, while nonlinear constraints make local search difficult, linear ones that are obtained from the reformulation–convexification techniques used in BARON will considerably benefit local optimization.

References

1. Ahmed, S., Tawarmalani, M. and Sahinidis, N.V. (2004), A finite branch-and-bound algorithm for two-stage stochastic integer programs, *Mathematical Programming*. 100, 355–377.
2. Ben-Tal, A., Eiger, G. and Gershovitz, V. (1994), Global minimization by reducing the duality gap, *Mathematical Programming* 63, 193–212.
3. Bisschop, J. and Meeraus, A. (1982), On the development of a general algebraic modeling system in a strategic planning environment, *Mathematical Programming Study* 20, 1–29.
4. Du, D., Gu, J. and Pardalos, P.M. (eds.), (1997), *Satisfiability Problem: Theory and Applications, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, Vol. 35, American Mathematical Society, Providence, RI.
5. Falk, J.E. and Soland, R.M. (1969), An algorithm for separable nonconvex programming problems, *Management Science* 15, 550–569.
6. Fourer, R., Gay, D. and Kernigham, B.W. (1993), *AMPL: A Modeling Language for Mathematical Programming,* The Scientific Press, San Francisco, CA.
7. Gau, T. and Schrage, L.E. (2003), Implementing a global solver in a general purpose callable library, *Global Optimization Theory Institute,* Argonne National Laboratory, September 8–10.
8. Ghildyal, V. (1997), Design and Development of a Global Optimization System. Master's thesis, Department of Mechanical & Industrial Engineering, University of Illinois, Urbana, IL.
9. Ghildyal, V. and Sahinidis, N.V. (2001), Solving global optimization problems with BARON. In: Migdalas, A., Pardalos, P. and Varbrand, P. (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 pp. 205–230.
10. Gu, J., Purdom, P.W., Franco, J. and Wah, B.W. (1997), Algorithms for the satisfiability (SAT) problem: A survey. In: Du, D., Gu, J. and Pardalos, P.M. (eds.), *Satisfiability Problem: Theory and Applications, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, Vol. 35, Providence, RI, pp. 19–151.
11. Hoos, H.H. and Stützle, T. (2000), SATLIB: An Online Resource for Research on SAT. In: Gent, I.P., van Maaren, H. and Walsh, T. (eds.), *Satisfiability Problem: Theory and Applications,* IOS Press, Amsterdam, pp. 283–292. SATLIB is available online at http://www.satlib.org/.
12. Horst, R. and Tuy, H. (1996), *Global Optimization, Deterministic Approaches,* third edition, Springer Verlag, Berlin.
13. Kearfott, R.B. (1996), *Rigorous Global Search: Continuous Problems, In: Nonconvex Optimization and Its Applications,* Vol. 13. Kluwer Academic Publishers, Dordrecht.
14. McCormick, G.P. (1972), Converting general nonlinear programming problems to separable nonlinear programming problems. Technical Report T-267, The George Washington University, Washington, D.C.

15. McCormick, G.P. (1976), Computability of global solutions to factorable non-convex programs: Part I – Convex underestimating problems, *Mathematical Programming* 10, 147–175.
16. McCormick, G.P. (1983), *Nonlinear Programming: Theory, Algorithms and Applications*, John Wiley & Sons.
17. Moore, R. (1993), *Interval Analysis,* Prentice Hall, Englewood Cliffs, NJ.
18. Quesada, I. and Grossmann, I.E. (1995), Global optimization of bilinear process networks and multicomponent flows, *Computers and Chemical Engineering* 19(12), 1219–1242.
19. Ryoo, H.S. and Sahinidis, N.V. (1995), Global optimization of nonconvex NLPs and MINLPs with applications in process design, *Computers and Chemical Engineering* 19, 551–566.
20. Ryoo, H.S. and Sahinidis, N.V. (1996), A branch-and-reduce approach to global optimization, *Journal of Global Optimization* 8, 107–139.
21. Sahinidis, N.V. (1996), BARON: A general purpose global optimization software package, *Journal of Global Optimization* 8, 201–205.
22. Sahinidis, N.V. (2003), Global optimization and constraint satisfaction: The branch-and-reduce approach, In: Bliek, A.C. Jermann, C. and Neumaier, A. (eds.), *Global Optimization and Constraint Satisfaction*, Lecture Notes in Computer Science Vol. 2861, Springer, Berlin, pp. 1–16.
23. Schichl, H., Dallwig, S. and Neumaier, A. (2001), The NOP-2 modeling language for nonlinear programming, *Annals of Operations Research* 104, 281–312.
24. Schweiger, C.A. and Floudas, C.A. (1998), MINOPT: A Modeling Language and Algorithmic Framework for Linear, Mixed-Integer, Nonlinear, Dynamic, and Mixed-Integer Nonlinear Optimization, Version 3.1, User's Manual'. Available at `http://titan.princeton.edu/MINOPT/minopt.html`.
25. Shectman, J.P. and Sahinidis, N.V. (1998), A finite algorithm for global minimization of separable concave programs, *Journal of Global Optimization* 12, 1–36.
26. Sherali, H.D. and Adams, W.P. (1999), *A Reformulation–Linearization Technique for Solving Discrete and Continuous Nonconvex Problems,* In: *Nonconvex Optimization and its Applications,* Vol. 3.1 Kluwer Academic Publishers, Dordrecht.
27. Sherali, H.D., Adams, W.P. and Driscoll, P.J. (1999), Exploiting special structures in constructing a hierarchy of relaxations for 0–1 mixed integer programs, *Operations Research* 46, 396–405.
28. Sherali, H.D. and Smith, J.C. (2001), Improving discrete model representations via symmetry considerations, *Management Science* 47, 1396–1407.
29. Sherali, H.D. and Tuncbilek, C.H. (1995), A reformulation–convexification approach for solving nonconvex quadratic programming problems, *Journal of Global Optimization* **7**, 1–31.
30. Sherali, H.D. and Wang, H. (2001), Global optimization of nonconvex factorable programming problems, *Mathematical Programming* 89, 459–478.
31. Smith, E.M.B. and Pantelides, C.C. (1996), Global optimisation of general process models. In: Grossmann I.E. (ed.), *Global Optimization in Engineering Design,* Kluwer Academic Publishers, Boston, MA, pp. 355–386.
32. Tawarmalani, M. and Sahinidis, N.V. (2001), Semidefinite relaxations of fractional programs via novel techniques for constructing convex envelopes of nonlinear functions, *Journal of Global Optimization* 20, 137–158.
33. Tawarmalani, M. and Sahinidis, N.V. (2002a), Convex extensions and convex envelopes of l.s.c. functions, *Mathematical Programming* 93, 247–263.
34. Tawarmalani, M. and Sahinidis, N.V. (2002b), *Convexification and Global Optimization in Continuous and Mixed-Integer Nonlinear Programming: Theory, Algorithms, Software, and*

Applications, Vol. 65 of *Nonconvex Optimization and Its Applications*, Kluwer Academic Publishers, Dordrecht.

35. Tawarmalani, M. and Sahinidis, N.V. (2004), Global optimization of mixed-integer nonlinear programs: A theoretical and computational study, *Mathematical Programming*. 99, 563–591.
36. Van Hentenryck, P., Michel, L. and Deville, Y. (1997), *Numerica: A Modeling Language for Global Optimization,* The MIT Press, Cambridge, MA.