BARON

Branch And Reduce Optimization Navigator

User's Manual
Version 4.0

Nikolaos V. Sahinidis

University of Illinois at Urbana-Champaign
Department of Chemical Engineering
600 South Mathews Avenue
Urbana, Illinois 61801, USA

`http://archimedes.scs.uiuc.edu`
E-mail: `nikos@uiuc.edu`
Phone: 217-244-1304
Fax: 217-333-5052

Version 1.0: 2 March 1995
Version 4.0: 29 March 1999
This revision: 28 June 2000

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Optimization problems with multiple local optima are encountered in all areas of engineering and the sciences. Determining global optima finds numerous applications in fields such as structural and shape optimization, mechanical equipment and parts design, analysis and design of control systems, integrated circuit design, prediction of molecular structures and molecular design, and chemical process synthesis and operations.

Motivated by a large number of potential applications, we have developed BARON, a computational system for facilitating the solution of nonconvex optimization problems to global optimality. The Branch And Reduce Optimization Navigator derives its name from its combining interval analysis and duality in its "reduce" arsenal with enhanced "branch and bound" concepts as it winds its way through the hills and valleys of complex optimization problems in search of global solutions.

1.1 Historical Notes

The first version of BARON was merely 1800 lines of code written in the GAMS modeling language [2] in 1991-93 when duality-based range reduction techniques were developed [16]. The code was initially applied to standard engineering design problems [16], design of just-in-time manufacturing systems [8], circuit layout and compaction [4], and chemical process planning [10].

The second version was approximately 10,000 lines of code written in FORTRAN 77 in 1994-95. This code incorporated some additional range reduction techniques [17] and was applied to polynomial and multiplicative programs [17], and robust controller design [22]. This code also included

specialized algorithms for separable concave minimization problems [19], and fixed-charge and other concave minimization problems arising in multiperiod planning problems [11]. At that time, heuristic techniques were added for feasibility-based range reduction as well as branching schemes to ensure finiteness for certain problem classes [19]. These specialized codes and FORTRAN version of BARON were first made available through anonymous FTP on 2 March, 1995.

Version 3.0 of BARON was developed in 1996-97. It offered, among other new features, more efficient memory management techniques, a specialized code for factorable nonlinear programming problems, an easy-to-use parser, and a detailed manual. This version of the code was approximately 23,000 lines of FORTRAN 90 code and 10,000 lines of code written in C.

Version 4.0 of the code was developed in 1997-98. Compressed data storage and fast tree traversal techniques were incorporated. The factorable NLP module allows an entirely linear programming based solution approach and utilizes faster gradient evaluation. The separable convex quadratic and indefinite quadratic solver were merged into a single module capable of handling mixed-integer positive definite, negative definite, and indefinite objectives. Modules can now interface to the linear programming solver CPLEX and the nonlinear programming solver SNOPT in addition to OSL and MINOS which previously were the only ones allowable. This code was used to solve blending and pooling problems [1], fractional programs in $0-1$ variables [20, 21], and multiplicative programs [18]. This version of BARON is currently approximately 26,000 lines of FORTRAN 90 code and 17,000 lines of code written in C.

1.2 Disclaimer

The program is provided with no warranty. It may be used and distributed freely as long as it is not sold for profit or incorporated in commercial products. Please report any comments and suggestions to Nick Sahinidis (`nikos@uiuc.edu`).

1.3 Acknowledgments

All my students contributed, each one in his own unique way, to the shaping, development and testing of the ideas that are implemented in this code. These students were, in chronological order, Russ Vander Wiel, Ming Long Liu, Hong Ryoo, Joseph Shectman, Ramon Gutierrez, Shabbir Ahmed,

Vinay Ghildyal, Mohit Tawarmalani, Nilanjan Adhya, Minrui Yu, Kevin Furman, Anastasia Vaia, Yannis Voudouris, Hussain Arsiwalla, Gautam Nanda, Mayank Mishra, and Sumit Mehra. Their contributions are far too many to be detailed. I am particularly indebted to Mohit.

I am also thankful to Michael Overton and Matthew Wilkins of New York University for proposing and helping with the development of the mixed-integer semidefinite programming module.

Chapter 2

Algorithmic Overview

We consider the problem of finding global solutions to general nonlinear and mixed-integer nonlinear programs:

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

where $f : X \to \mathbb{R}$, $g : X \to \mathbb{R}^m$, and $X \subset \mathbb{R}^n$.

In solving P, we will rely on the existence of another optimization problem, R, the "relaxation" or "relaxed problem," whose optimal solution provides a lower bound on the solution of P. This relaxation is constructed by enlarging the feasible region and/or underestimating the objective function of P:

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

where $\bar{f} : \bar{X} \to \mathbb{R}$, $\bar{g} : \bar{X} \to \mathbb{R}^{\bar{m}}$, $\bar{X} \subset \mathbb{R}^{\bar{n}}$, and for any x feasible to P, there exists $\bar{x}$ feasible to R with $\bar{f}(\bar{x}) \leq f(x)$. In many applications, the sets of variables or constraints are the same in problems P and R. In general, however, relaxations can be constructed in higher dimensional spaces than the original problem or even in an altogether different space. To simplify the notation, from now on we will drop the bars and use x, f, g to denote the problem variables, objective and constraint functions of both problems P and R; the corresponding optimization problem will be clear from the context.

As long as P has a finite-valued solution, it is always possible to construct a relaxation for it. To be precise, it is possible to construct infinitely many different relaxations. The discussion of techniques for the construction of relaxations will be deferred to Section 2.5. Typically, we will construct relaxations that are convex optimization problems so that their solution can be found through conventional linear or nonlinear programming techniques. All that we require, however, in the next section is that R be (much) easier to solve than P.

2.1 Branch and Bound

For most problems of interest, it is possible to construct relaxations in such a way so that the difference between the optimal objective function values of P and R is a nonincreasing function of the size of the feasible region of P. It is this property that makes possible the application of a branch and bound algorithm for solving P.

A graphical interpretation of branch and bound is shown in Figure 2.1 for a univariate nonconvex function with two local minima. Once the relaxed problem is solved, a valid lower bound, L, is obtained for the global minimum (Figure 2.1.a) of P. Using the relaxed solution as a starting point, if some other more advantageous starting point is not available, local minimization techniques can be used to obtain an upper bound, U, for the global solution of P (Figure 2.1.b). At this stage, the global minimum is known to be between L and U. If $U - L$ is sufficiently small, the algorithm terminates. Otherwise, branch and bound subdivides the feasible region into parts. In a typical approach, two parts are generated: one to the right and one to the left of the relaxed problem solution. A new relaxed problem is solved for each subdivision. This time, the relaxations represent a more accurate approximation of the original function and the least of their solutions provides the new lower bound. This bound is closer to the upper bound (Figure 2.1.c). The process of subdividing the domain and computing lower and upper bounds on the optimal objective over each subdivision is repeated until the lower and upper bounds become sufficiently close.

The entire branch and bound process may be represented on a tree whose nodes and branches correspond to solving relaxations and partitioning the search space, respectively (Figure 2.1.d). During the subdivision process, nodes of the search space whose lower bounds exceed the upper bound may be excluded from further consideration as they clearly do not include any superior solutions than the one already at hand. These nodes are termed as

"fathomed." Node $R2$ in Figure 2.1.d represents one such node.

A standard implementation branch and bound algorithm is illustrated in Figure 2.2. At a given iteration, a number of partition elements (open nodes) is available and maintained in a list. One of the open partitions is chosen and its corresponding relaxation is solved. If the lower bound indicates that the solved node is inferior than the current incumbent (best known solution of P), the node is abandoned and another node is selected. If the node cannot be discarded immediately, it is more finely subdivided into a set of new nodes that replace it on the list of open nodes; a process known as branching. The process of selecting a node, lower bounding and branching is repeated until the list of open nodes becomes empty.

2.2 Optimality-Based Range Reduction

Consider the perturbation of problem R:

$$(R_y): \qquad \varphi(y) = \begin{array}{ll} \min & f(x) \\ \text{s.t.} & g(x) \leq y \\ & x \in X \end{array}$$

If R is convex, so is R_y for any y, and $\varphi(y)$ is a convex function. Assume now that the solution of R has a value of L, *i.e.*, $\varphi(0) = L$, and that the constraint $x_j - x_j^U \leq 0$ is active at this solution. The value function $\varphi(y)$ is depicted in bold line in Figure 2.3 where perturbations are considered only with respect to the right hand side of constraint $x_j - x_j^U \leq 0$. Let us now assume that a valid upper bound, U, for P is available. Clearly, the intersection of the value function and the line $\varphi(y) = U$ corresponds to a valid lower bound, κ_j^*, for x_j. In this case, κ_j^* is tighter than x_j^L, which was used to construct the relaxation in the first place.

Unfortunately, computing κ_j^* requires knowledge of the value function $\varphi(y)$, which is not explicitly given. Nonlinear programming duality comes to the rescue at this point as it provides an easily computable valid lower bound, $\kappa_j \leq \kappa_j^*$, for x_j in this context. It is well known that, if R has an optimum of finite value, $\lambda_j \in \mathbb{R}$ is a Lagrange multiplier for R, corresponding to constraint $x_j - x_j^U \leq 0$, if and only if the hyperplane with equation $z = \varphi(0) - \lambda_j y$ is a supporting hyperplane at $y = 0$ of the graph of the perturbation function $\varphi(y)$. In other words, λ_j is a Lagrange multiplier if and only if

$$\forall y \in \mathbb{R} : \varphi(y) \geq \varphi(0) - \lambda_j y.$$

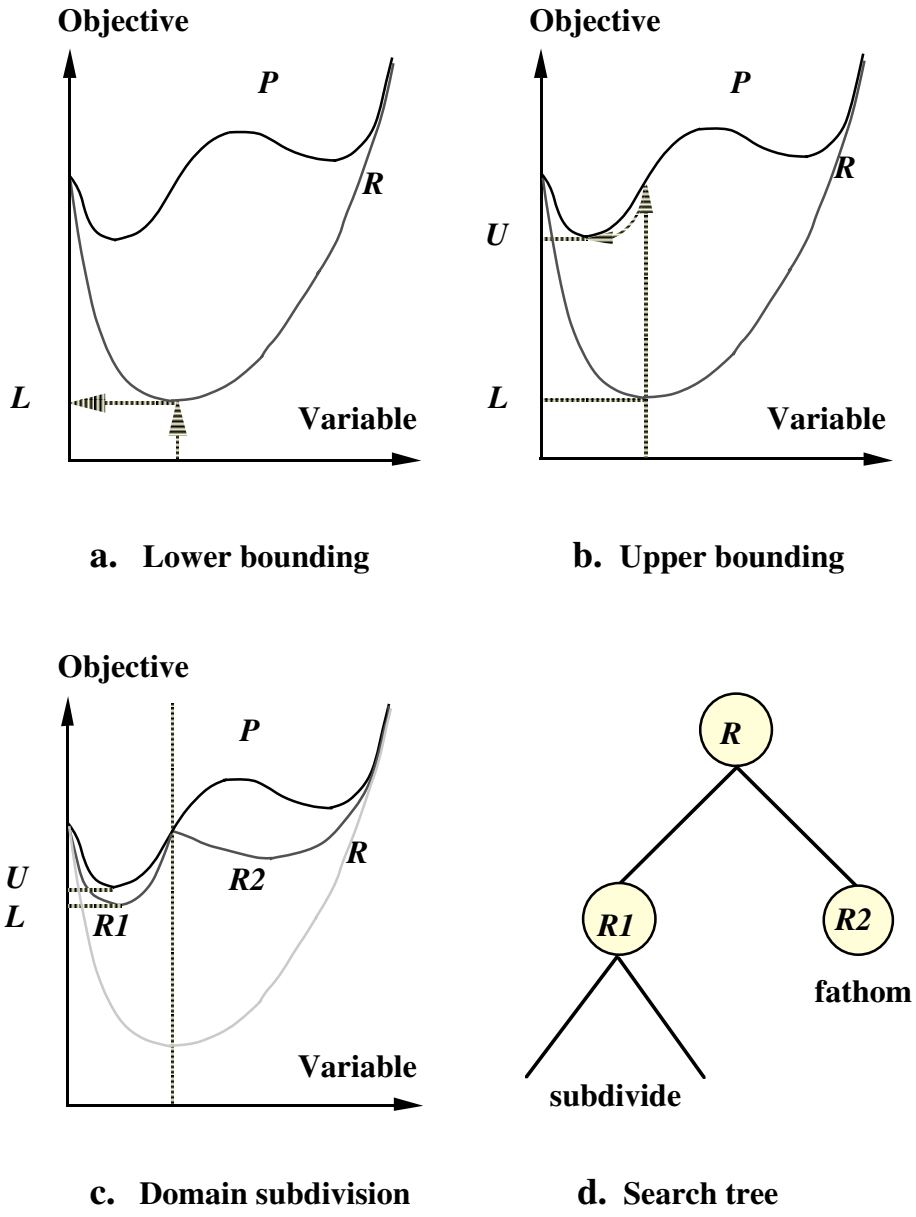

Figure 2.1: The principles of branch and bound.

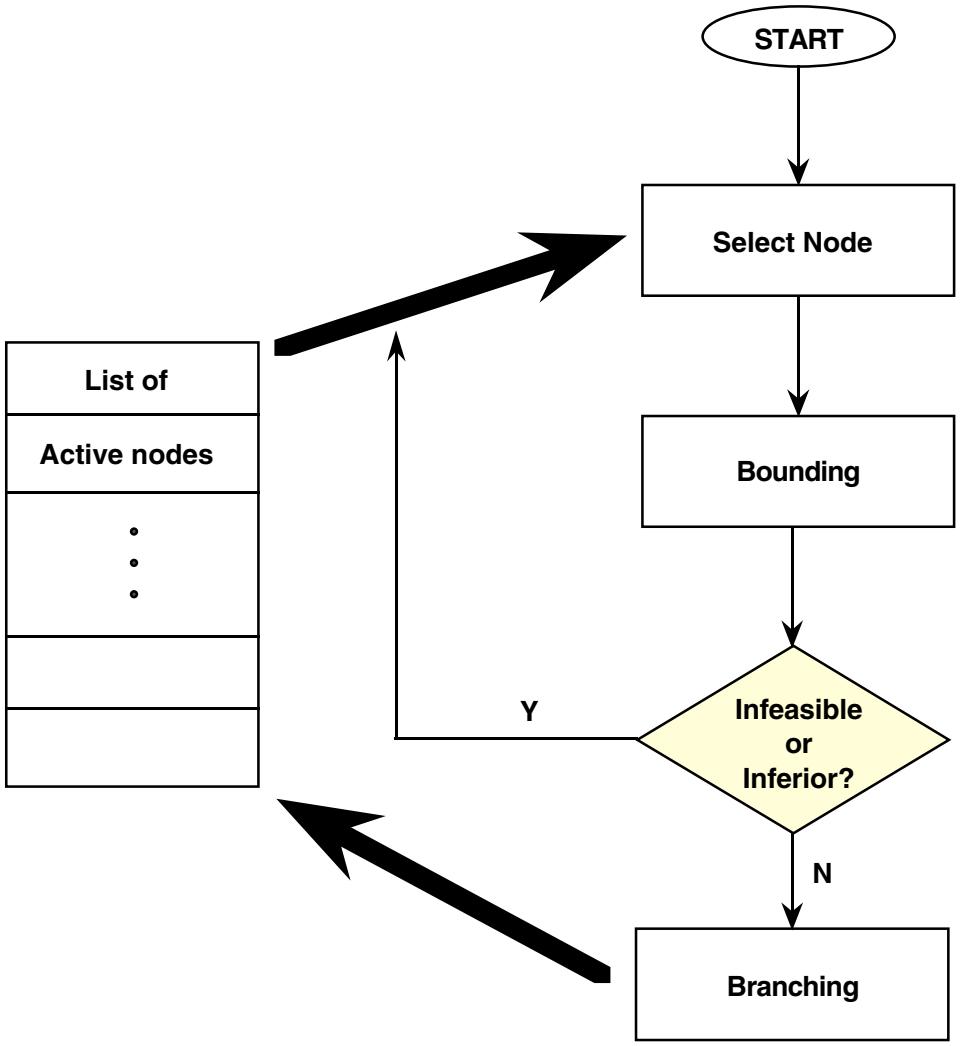

Figure 2.2: Branch and bound algorithm.

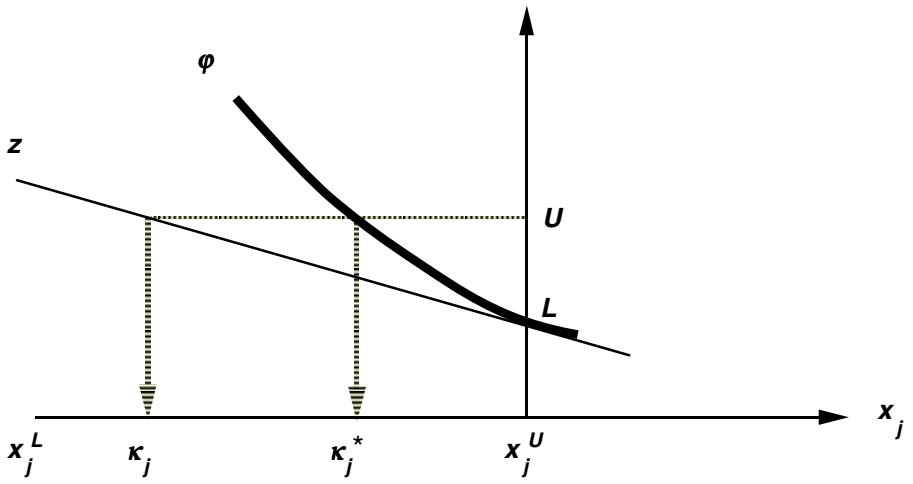

Figure 2.3: Range reduction using marginals.

Utilizing this property, we can find a valid lower bound for x_j simply by computing the intersection of the supporting hyperplane z and the line $\varphi(y) = U$. Assuming $\lambda_j > 0$, the potentially improved lower bound for x_j is:

$$\kappa_j = x_j^U - (U - L)/\lambda_j.$$

It is not hard to see that if the range constraint $x_j^L \leq x_j$ is active at the solution of R, one can obtain a valid upper bound for x_j by an argument that is analogous to the one used in relation to Figure 2.3. If a variable, x_j, is at neither of its bounds in the relaxed problem solution, we can *probe* its bounds. Referring to Figure 2.4, we can temporarily fix this variable at its lower (upper) bound, construct the linear underestimator, $z^L(z^U)$, of the value function and obtain a valid range, $[\kappa_j, \pi_j]$, for the said variable from the intersection of the linear underestimators and $\varphi(y) = U$. As in the case of active constraints, knowledge of the entire value function produces tighter bounds $\kappa_j^* \geq \kappa_j$ and $\pi_j^* \leq \pi_j$. Unlike the case of active constraints, obtaining tighter bounds using probing requires the solution of additional relaxed problems when a variable is temporarily fixed at a bound.

The above process of range contraction can be extended to arbitrary constraints of the type $g_i(x) \leq 0$, or even to sets of constraints that may or may not be active at the relaxed problem solution. Some of the valid inequalities derived in this way and the range reduction mechanisms based

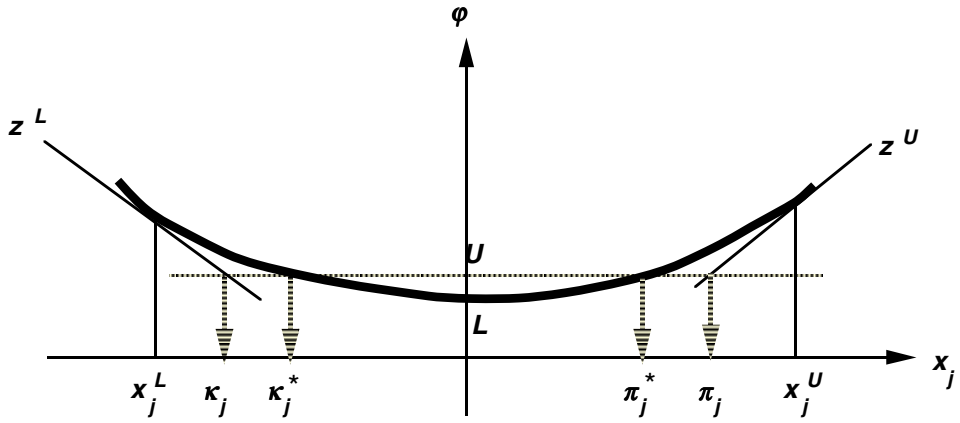

Figure 2.4: Range reduction using probing.

on them are summarized in Tables 2.1 and 2.2. We use μ to denote the optimal dual multipliers of linear/nonlinear constraints and λ to denote the optimal multipliers (reduced costs) of simple variable bounds (range constraints). L and Z in these tables denote the objective function values of the relaxed and probing subproblems, respectively.

These valid inequalities were derived based on the optimal solution of the relaxed problem and by using an optimality argument. For this reason, they will be referred to as *optimality-based* valid inequalities. Although they may exclude solutions that are feasible to P, they do not exclude any solutions of P with objective function values better than U. As these inequalities reduce the range of constraints and variables, they will be referred to as optimality-based *range reduction* or *range contraction* mechanisms.

Table 2.1: Range reduction derived from active constraints.

Active Constraint	Requirement	Valid Inequality
$g_i(x) \leq 0$	$\mu_i > 0$	$g_i(x) \geq -(U-L)/\mu_i$
$a_i^t x \leq b_i$	$\mu_i > 0$	$a_i^t x \geq b_i - (U-L)/\mu_i$
$x_j \leq x_j^U$	$\lambda_j > 0$	$x_j \geq x_j^U - (U-L)/\lambda_j$
$x_j^L \leq x_j$	$\lambda_j > 0$	$x_j \leq x_j^L + (U-L)/\lambda_j$

Table 2.2: Range reduction derived from inactive constraints after probing.

Inactive Constraint	Requirement	Valid Inequality
$a_i^t x \leq b_i$	Add $b_i \leq a_i^t x$ to R. Solve R and obtain Z. $\mu_i > 0$.	$a_i^t x \leq b_i + (U - Z)/\mu_i$.
$x_j \leq x_j^U$	Add $x_j^U \leq x_j$ to R. Solve R and obtain Z. $\lambda_j > 0$.	$x_j \leq x_j^U + (U - Z)/\lambda_j$
$x_j^L \leq x_j$	Add $x_j \leq x_j^L$ to R. Solve R and obtain Z. $\lambda_j > 0$.	$x_j \geq x_j^L - (U - Z)/\lambda_j$

2.3 Feasibility-Based Range Reduction

Feasibility-based tightening, or feasibility-based range reduction, is a process that generates constraints that cut-off infeasible portions of the solution space.

Consider, for example, the constraints $\sum_{j=1}^{n} a_{ij} x_j \leq b_i$, $i = 1, ..., m$. Then, one of the constraints

$$\begin{cases} x_h \leq \frac{1}{a_{ih}} \left(b_i - \sum_{j \neq h} \min \left\{ a_{ij} x_j^U, a_{ij} x_j^L \right\} \right), & a_{ih} > 0 \\ x_h \geq \frac{1}{a_{ih}} \left(b_i - \sum_{j \neq h} \min \left\{ a_{ij} x_j^U, a_{ij} x_j^L \right\} \right), & a_{ih} < 0 \end{cases} \tag{2.1}$$

is also valid for each pair (i, h) that satisfies $a_{ih} \neq 0$.

Of course, to tighten variable bounds based on the above linear constraints, one could simply solve the $2n$ LPs:

$$\left\{ \min \pm x_k \text{ s.t. } \sum_{j=1}^{n} a_{ij} x_j \leq b_i, i = 1, \ldots, m \right\}, k = 1, \ldots, n, \tag{2.2}$$

which would provide tightening that is optimal, albeit computationally expensive. In this regard, the latter cuts 2.1 function as "poor man's linear programs," particularly when they are applied iteratively, looping over the set of variables several times.

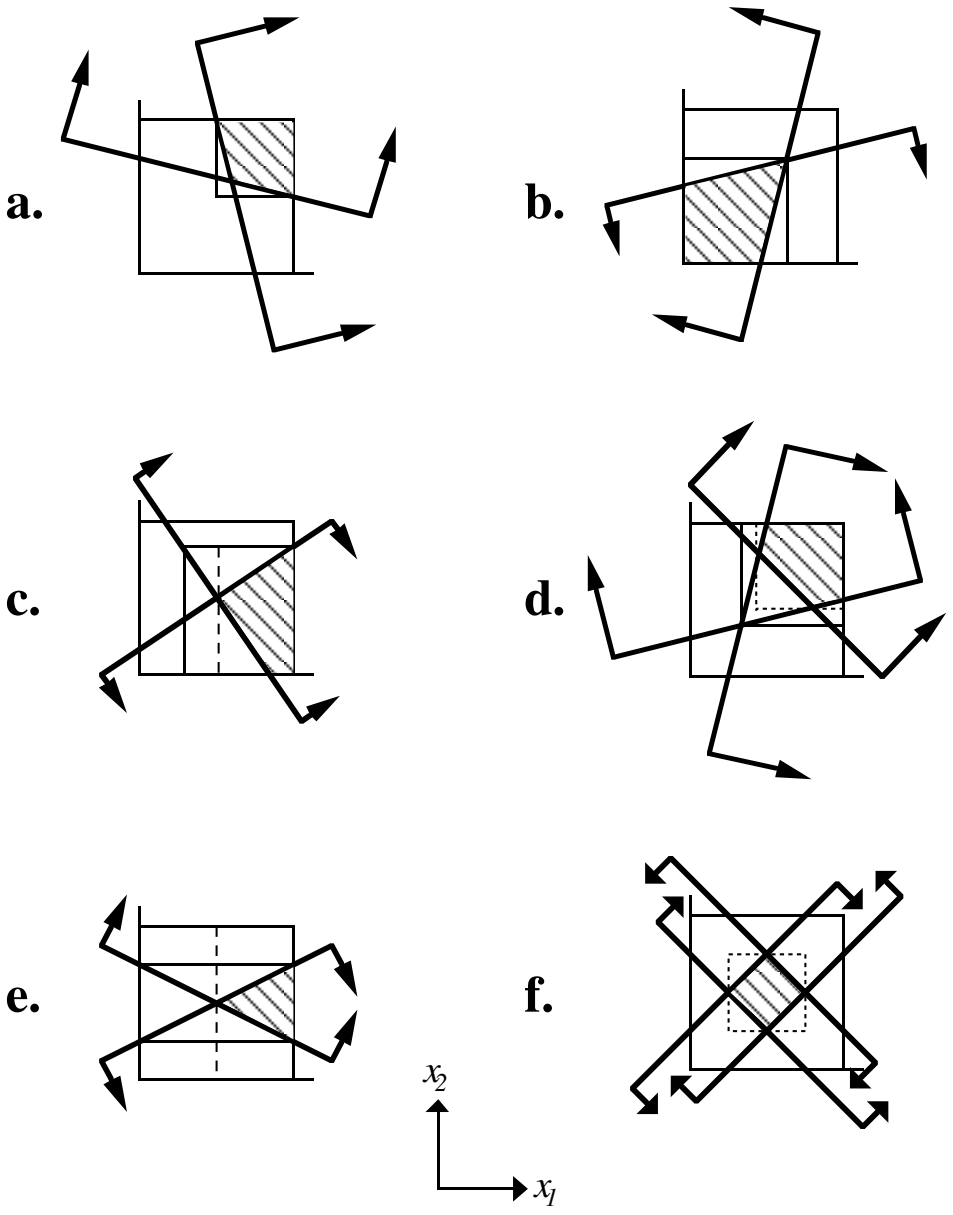

Figure 2.5: Poor man's LPs.

Figure 2.5 shows how an implementation of 2.1 compares to the solution of the LPs for different two-dimensional examples. In each instance, the outer box represents the bound set before tightening begins, with constraints shown in bold lines and the feasible region shaded. Bounds improved by 2.1 are shown in plain line; improvements by 2.2 are shown in dashed line, when they differ from those of 2.1.

In Figures 2.5.a and 2.5.b, techniques 2.1 and 2.2 give the same result. In Figure 2.5.c, the effects of 2.1 agree with the effect of 2.2 for variable x_2, while also improving the bounds on x_1, albeit not to the maximum possible extent. In Figure 2.5.d, 2.1 improves bounds on both variables, although neither bound is improved to the maximum possible extent. Figure 2.5.e shows the bounds on x_2 tightened to their full extent by 2.1, but here the heuristic fails to improve bounds on x_1, at all. Figure 2.5.f is particularly insightful as a pathological case for the heuristic. In the latter case, the bounds are not improved at all, whereas a great deal of bounds reduction is possible, illustrated by the four LP solutions (dashed lines).

One can see why Figure 2.5.f is pathological for the "poor man's linear programs." The heuristic can make use of only one bound and one constraint at a time, while linear programming considers the entire constraint set simultaneously. In practice, a case such as Figure 2.5.f would not occur in the context of branch and bound if all of the $2n$ LPs are solved initially, in preprocessing, on a one-time basis. Thereafter, for each subdomain, at least one bound acts as an nonredundant constraint. Finally, note that sometimes the heuristic achieves its maximum domain reduction asymptotically, *e.g.*, Figure 2.5.b and Figure 2.5.d, where improved bounds on variable x_1 then enable bound improvement on variable x_2 that, in turn, facilitate further tightening of x_1 on the next pass, etc.

2.4 Branch and *Reduce*

The range reduction techniques of the previous section can be employed to preprocess a global optimization problem before the application of any global optimization algorithm. In the context of a branch and bound algorithm, range reduction can be used to improve the performance of the bounding procedure at *every* node of the search tree. As the emphasis is on range reduction, we refer to the resulting algorithm as a branch and *reduce* algorithm.

For a given problem, BARON's global optimization strategy integrates conventional branch and bound with a wide variety of range reduction tests

(Figure 2.6). These tests are applied to every subproblem of the search tree in pre- and post-processing steps to contract the search space and reduce the relaxation gap. Many of the reduction tests are based on duality and are applied when the relaxation is convex and solved by an algorithm that provides the dual, in addition to the primal, solution of the relaxed problem. Another crucial component of the software is the implementation of heuristic techniques for the approximate solution of optimization problems that yield improved bounds for the problem variables (feasibility-based tightening).

Finally, the algorithm incorporates a number of compound branching schemes that accelerate convergence of standard branching strategies as outlined in Section 2.6.

2.5 Lower Bounding

Typically, the relaxed problem is constructed using factorable programming techniques (McCormick, [12], [13], [14]), so that the relaxations are exact at the variable bounds. The tightness of the relaxation depends on the tightness of the variable bounds. Construction of underestimators for certain nonconvex terms is described below.

1. Bilinearities of the form $x_i x_j$ can be underestimated by two constraints, depending on the sign of the inequality in which they appear and the sign of the bilinear term. If the inequality is of the type $x_i x_j + g(x) \leq 0$, the following constraints may be used:

$$\begin{aligned} w_{ij} + g(x) &\leq 0 \\ w_{ij} &\geq x_j^U x_i + x_i^U x_j - x_i^U x_j^U \\ w_{ij} &\geq x_j^L x_i + x_i^L x_j - x_i^L x_j^L \end{aligned}$$

   where, in order to maintain differentiability, a new variable, w_{ij}, was substituted for $x_i x_j$.

   If the inequality in which bilinear terms appear is of the type $x_i x_j + g(x) \geq 0$, the following constraints may be employed:

$$\begin{aligned} w_{ij} + g(x) &\geq 0 \\ w_{ij} &\leq x_j^L x_i + x_i^U x_j - x_i^L x_j^U \\ w_{ij} &\leq x_j^U x_i + x_i^L x_j - x_i^U x_j^L \end{aligned}$$

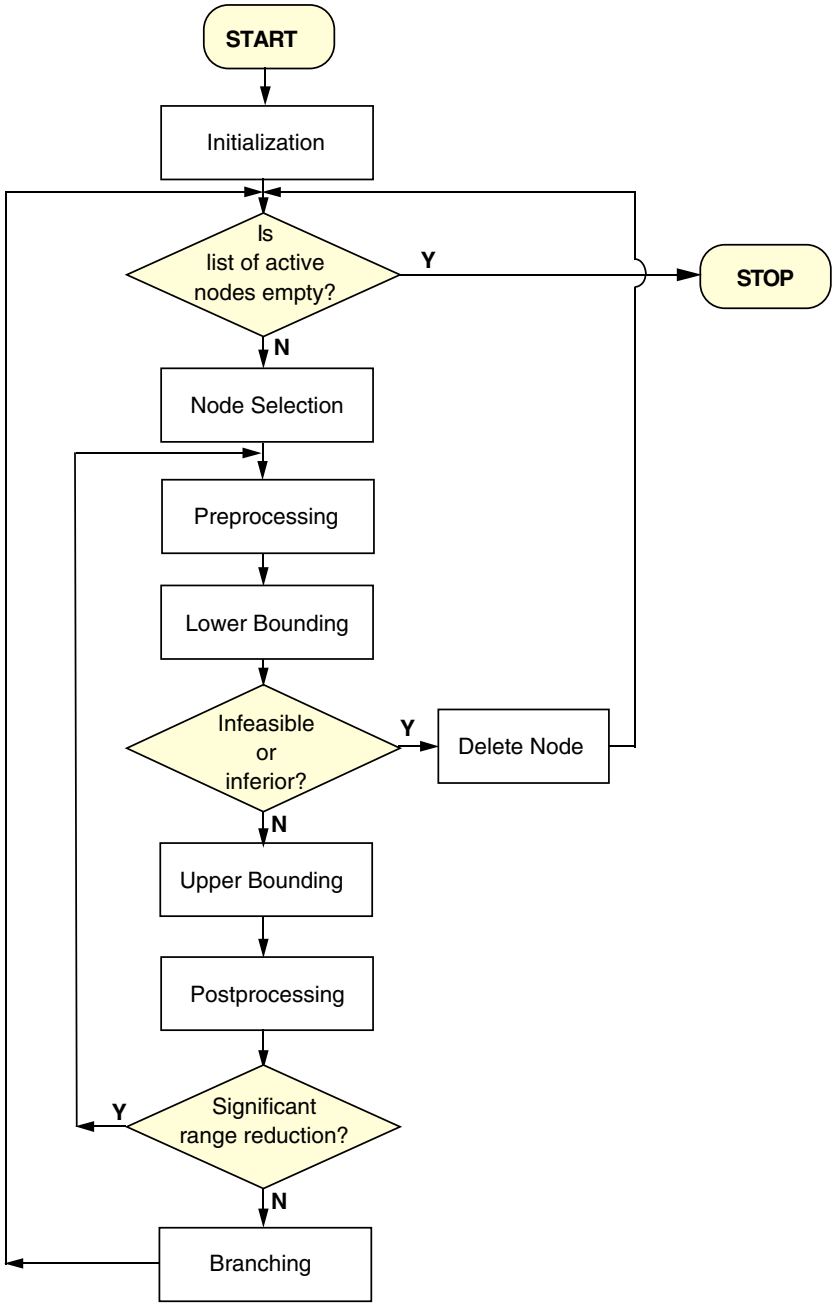

Figure 2.6: Branch and reduce algorithm.

2. Similarly, linear fractional terms of the form $\frac{x_i}{x_j}$, with $x_j \geq x_j^L > 0$, can also be underestimated by either the first or last two of the following four constraints, depending on the sign of the inequality in which they appear:

$$\frac{x_i}{x_j} \geq \frac{x_i^U}{x_j} + \frac{x_i}{x_j^L} - \frac{x_i^U}{x_j^L}$$
$$\frac{x_i}{x_j} \geq \frac{x_i^L}{x_j} + \frac{x_i}{x_j^U} - \frac{x_i^L}{x_j^U}$$
$$\frac{x_i}{x_j} \leq \frac{x_i^U}{x_j} + \frac{x_i}{x_j^U} - \frac{x_i^U}{x_j^U}$$
$$\frac{x_i}{x_j} \leq \frac{x_i^L}{x_j} + \frac{x_i}{x_j^L} - \frac{x_i^L}{x_j^L}$$

3. Univariate terms which are concave over their entire domain can be underestimated by their secant. Such concave terms include x^n with $0 \leq n \leq 1, x \geq 0$ and x^n with $n = 2k+1, k \in \{1, 2, \ldots\}, x \leq 0$. Consider any such function, $g(x), x \in [l, u]$. This concave function can be underestimated by the term $\alpha x + \beta$, where

$$\alpha = \frac{g(u) - g(l)}{u - l},$$
$$\beta = g(u) - \alpha u.$$

4. Terms of the form x^n where n is odd and $0 \in [l, u]$, can be treated in two different ways.

   (a) The interval $[l, u]$ can be partitioned at $x = 0$ resulting into two subproblems such that x^n is convex in one subproblem and concave in the other. The concave part can then be treated as above. This process increases the number of open nodes in the branch and bound tree.

   (b) If branching is not desired, a convex envelope, $\gamma(x)$, of the term can be obtained as shown in Figure 2.7:

$$\gamma(x) = \begin{cases} x_n, & \text{if } x \geq \xi, \\ \alpha x + \beta, & \text{if } x < \xi. \end{cases}$$

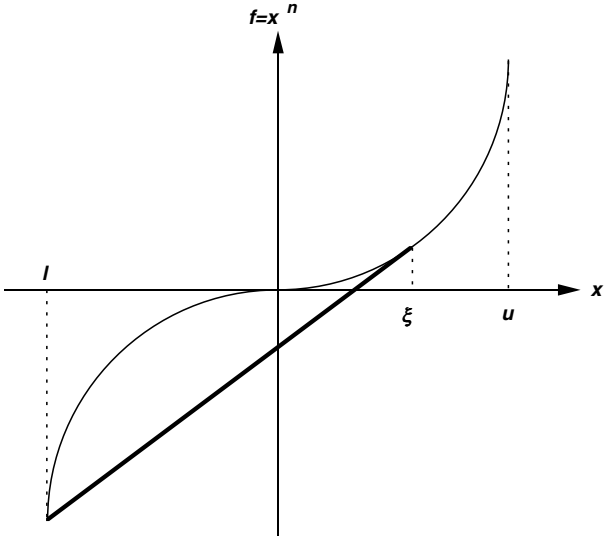

Figure 2.7: Monomial underestimator.

The point ξ is where the monomial and its underestimator have the same slope and is, therefore, determined by solving the equation, $(n-1)\xi^n - nl\xi^{n-1} + l^n = 0$. The coefficients α and β are then given by:

$$\alpha = n\xi^{n-1},$$
$$\beta = l^n - \alpha l.$$

2.6 Branching

Here, the domain of the current node is partitioned into a finite number of subregions through a *rectangular* subdivision scheme. This involves selecting a branching variable and a branching point in its domain. Typically, the branching variable is the one which "contributes" the most to the relaxation gap, *i.e.*, the one with the largest "violation." Measures of the violation include the distance between the underestimator and the nonconvex term at the solution of the relaxation, the ranges of the variables and the distance of the nearest variable bound from its solution value. In most cases, the violations are calculated by combining all these criteria.

Once the branching variable is identified, the branching point is chosen as follows:

1. Once every predetermined number of branch and bound iterations, the variable is branched at its midpoint (*bisection*). This ensures a significant reduction in the domains of the created subproblems and hence *exhaustiveness*, which is a key to finiteness, at least for certain problem classes [19].

2. If the incumbent solution is present in the current subdomain, the value of the incumbent is used as the branching point provided that this process leads to two new subproblems that are distinct from their father node. This partitioning rule makes the incumbent gapless and is also key to finiteness.

3. If the above two conditions do not apply, the branching variable is branched at the solution of the lower bounding problem (ω-branching).

At the end of the branching step, two subproblems are created and a new node is selected for preprocessing. The branch and reduce algorithm is now illustrated through an example.

2.7 Illustrative Example 1

The branch and bound algorithm for continuous global optimization is demonstrated on the following problem:

$$\text{(P)} \qquad \begin{aligned} z_{\text{opt}} = \min \quad & -x_1 - x_2 \\ \text{s.t.} \quad & x_1 x_2 \leq 4 \\ & 0 \leq x_1 \leq 6 \\ & 0 \leq x_2 \leq 4 \end{aligned}$$

See Figure 2.8 for a graphical illustration of the problem. The problem exhibits two local minima at points A and B with objective function values of -5 and $-6\frac{2}{3}$, respectively. Point B is thus the global minimum we seek through the branch and bound algorithm.

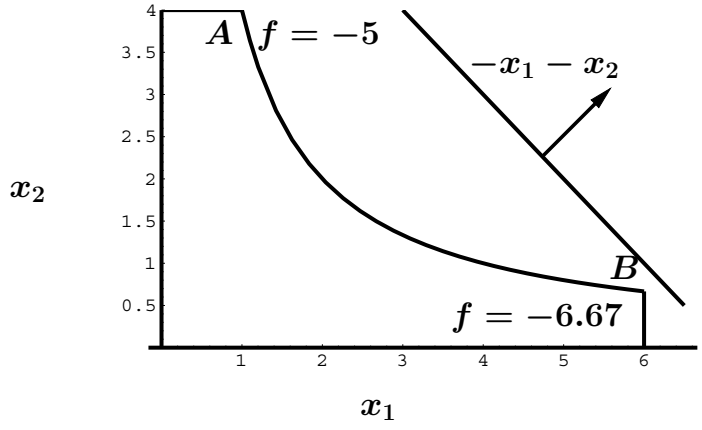

Figure 2.8: Feasible region for Example 1

2.7.1 A separable relaxation

The feasible region of (P) is nonconvex and the objective function is linear (convex). We construct a convex relaxation of (P) by outer-approximating its feasible set with a convex set by using a separable reformulation scheme which employs the following alegebraic identity:

$$x_1 x_2 = \frac{1}{2}\left\{(x_1 + x_2)^2 - x_1^2 - x_2^2\right\}$$

Amongst the three square terms on the right hand side of the above equation, $(x_1 + x_2)^2$ is convex while $-x_1^2$ and $-x_2^2$ are concave. We relax $-x_1^2$ and $-x_2^2$

by linear underestimators using the starting bounds for x_1 ($[0,6]$) and x_2 ($[0,4]$), respectively. The following relaxation of (P) results:

$$\text{(R)} \qquad \begin{aligned} z_{\text{opt}} \geq \min \quad & -x_1 - x_2 \\ \text{s.t.} \quad & (x_1 + x_2)^2 - 6x_1 - 4x_2 \leq 8 \\ & 0 \leq x_1 \leq 6 \\ & 0 \leq x_2 \leq 4. \end{aligned}$$

The relaxation (R) is depicted in 2.9.

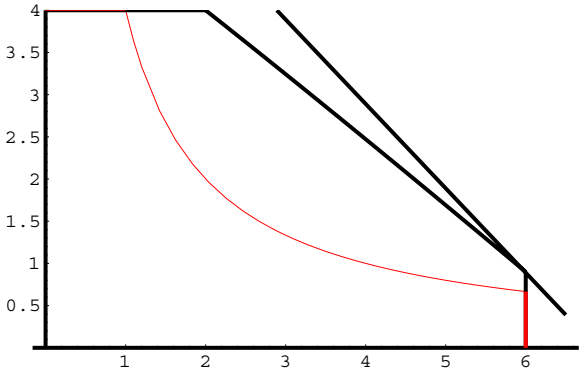

Figure 2.9: Separable Relaxation

An alternate, and in fact tighter, relaxation for (P) can be derived using the convex envelope of x_1x_2 over the rectangle $[0,6] \times [0,4]$:

$$xy \geq \max \left\langle \begin{array}{c} x^U y + y^U x - x^U y^U \\ x^L y + y^L x - x^L y^L \end{array} \right\rangle .$$

We choose the separable relaxation since it results in a more interesting branch and bound tree for this example and allows us to demonstrate of the effect of acceleration techniques for a branch and bound algorithm. The factorable programming relaxation produces the convex hull for (P) and therefore terminates with the optimal solution at the root node of the tree as we shall detail later.

Root Node: The optimal solution for (R) is $(x_1, x_2) \approx (6, 0.89)$ with an objective function value of -6.89. When a local search is performed on (P) starting at the relaxation solution $(6, 0.89)$, the point B, $(6, 2/3)$, is obtained with an objective function value of $-6\frac{2}{3}$. Thus, we have shown that the objective function value corresponding to the global optimum lies

in the interval

$$z_{\text{opt}} \in \left[-6.89, -6\frac{2}{3}\right].$$

Branching Variable: The optimal solution for (R) has x_1 at its upper bound. It is thus not wise to branch on x_1 for generating the partitions since x_1 does not contribute for any violations in the relaxation. We choose x_2 as the branching variable. $x_2 = 1$ is chosen as the branching point since it bisects the feasible interval for x_2. The resulting variable bounds are shown in Figure 2.10.

Node Selection: We choose the left child of the root node for further exploration. For this node, $x_1 \in [0, 6]$ and $x_2 \in [0, 2]$. We update our relaxation to take advantage of these bounds:

$$(\text{R}_L) \qquad \begin{aligned} z_{\text{opt}} \geq \min \quad & -x_1 - x_2 \\ \text{s.t.} \quad & (x_1 + x_2)^2 - 6x_1 - 2x_2 \leq 8 \\ & 0 \leq x_1 \leq 6 \\ & 0 \leq x_2 \leq 2 \end{aligned}$$

Solving the relaxation, we obtain a lower bound of -6.74. We partition this node further at $x_2 = 1$.

Node Selection: We choose the right child of the root node for further exploration. For this node, $x_1 \in [0, 6]$ and $x_2 \in [2, 4]$. We update our relaxation to take advantage of these bounds:

$$(\text{R}_R) \qquad \begin{aligned} z_{\text{opt}} \geq \min \quad & -x_1 - x_2 \\ \text{s.t.} \quad & (x_1 + x_2)^2 - 6x_1 - 6x_2 \leq 0 \\ & 0 \leq x_1 \leq 6 \\ & 2 \leq x_2 \leq 4 \end{aligned}$$

Solving the relaxation, we obtain a lower bound of -6.00. The best known solution has an objective function value of $-6\frac{2}{3}$. Therefore, the current node does not contain an optimal solution and can safely be fathomed.

Upon further exploration, we observe that the branch and bound algorithm proves optimality of $(6, 2/3)$ within an absolute tolerance of 0.01 after conducting 5 iterations. For a graphical representation of the relaxations, refer to Figure 2.11.

Next, we present the effect of acceleration techniques on the above example.

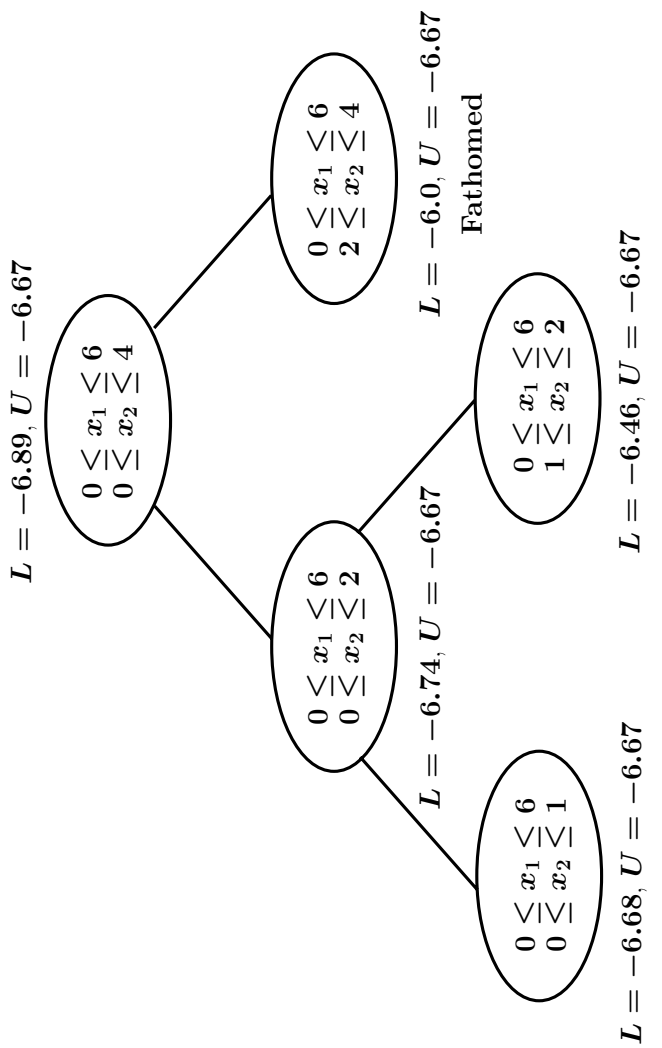

Figure 2.10: Branch and Bound tree for separable relaxation

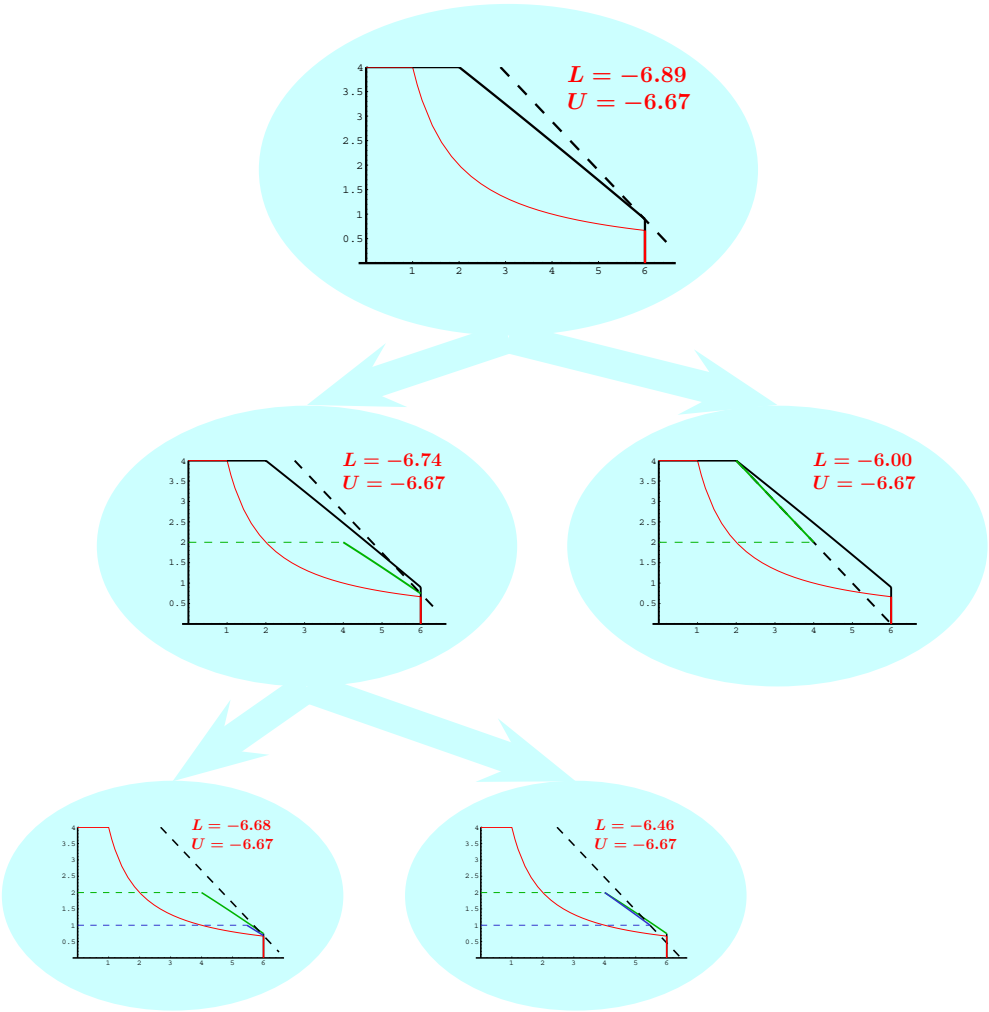

Figure 2.11: Separable relaxations generated for Example 1

2.7.2 Tighter Relaxation

If factorable programming techniques are used to relax x_1x_2, then (P) relaxes to:

$$\text{(BR)} \qquad \begin{array}{ll} \min & -x_1 - x_2 \\ \text{s.t.} & 4x_1 + 6x_2 \leq 28 \\ & 0 \leq x_1 \leq 6 \\ & 0 \leq x_2 \leq 4 \end{array}$$

The optimum for (BR) lies at $(6, 2/3)$, which also provides an upper bound for the problem. Therefore, the problem is solved exactly at the root node. As shown in Figure 2.12, the factorable relaxation provides the convex hull of the nonconvex problem.

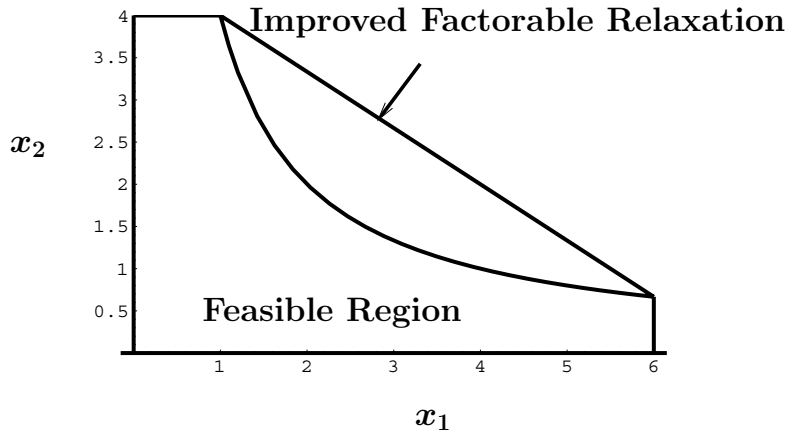

Figure 2.12: Factorable relaxation for Example 1

2.7.3 Optimality-Based Range-Reduction

In the solution of the root node relaxation, x_1 goes to its bound with a lagrange multiplier $\lambda_1 = 0.2$. Applying marginals-based range-reduction, it follows that the lower bound for x_1 may be updated to:

$$x_1^L = x^U - (U - L)/\lambda_1 \approx 6 - (-6.67 + 6.89)/0.2 \approx 4.86$$

By probing on the lower bound of x_2, we get

$$x_2^L = 0 + 6\frac{2}{3} - 6 = \frac{2}{3}.$$

Reconstructing the relaxation with the new bounds, we get:

$$\begin{array}{lll} \text{(MR)} & \min & -x_1 - x_2 \\ & \text{s.t.} & (x_1 + x_2)^2 - 10.86x_1 - 14/3x_2 + 23.82 <= 0 \\ & & 4.86 \leq x_1 \leq 6 \\ & & 0.66 \leq x_2 \leq 4 \end{array}$$

The resulting lower bound is -6.67 and thus global optimality is proven without branching even though the algorithm was based on a weak relaxation.

2.7.4 Branching on Incumbent

Once again, we consider the separable relaxation. This time, instead of branching the root node at $x_2 = 2$, we branch on the incumbent solution $x_2 = 2/3$. After branching, the relaxation at the left node is:

$$\begin{array}{lll} \text{(IR}_L\text{)} & \min & -x_1 - x_2 \\ & \text{s.t.} & (x_1 + x_2)^2 - 6x_1 - \dfrac{2}{3}x_2 \leq 8 \\ & & 0 \leq x_1 \leq 6 \\ & & 0 \leq x_2 \leq 2/3 \end{array}$$

The lower bound for IR_L is $-6\frac{2}{3}$ and the node is fathomed. For the right child of the root node, the relaxation is:

$$\begin{array}{lll} \text{(IR}_R\text{)} & \min & -x_1 - x_2 \\ & \text{s.t.} & (x_1 + x_2)^2 - 6x_1 - \dfrac{14}{3}x_2 \leq \dfrac{16}{3} \\ & & 0 \leq x_1 \leq 6 \\ & & 2/3 \leq x_2 \leq 4 \end{array}$$

Again, the optimal solution is found at $(6, 2/3)$ and the node is fathomed. The globality of the solution is proven in only three nodes whereas the standard branching scheme required five nodes.

2.8 Illustrative Example 2

Consider the following concave quadratic programming problem [5].

$$\min \quad 42x_1 - 50x_1^2 + 44x_2 - 50x_2^2 + 45x_3 - 50x_3^2$$

$$
\begin{aligned}
& +47x_4 - 50x_4^2 + 47.5x_5 - 50x_5^2 \\
\text{s.t.} \quad & 20x_1 + 12x_2 + 11x_3 + 7x_4 + 4x_5 \leq 40 \\
& 0 \leq x_i \leq 1, i = 1, \ldots, 5
\end{aligned}
$$

The global minimum for this problem occurs at $x = (1, 1, 0, 1, 0)$, with an objective function value of -17.

A standard branch-and-bound algorithm using bisection branching was first used to solve this problem. This algorithm requires 17 iterations for the upper and lower bounds to converge with a tolerance of 10^{-6}. The branch and reduce algorithm, applied to the same problem, requires only 7 iterations. The branch and reduce tree is outlined in Figure 2.13. At each node, L_i and U_i represent the lower and upper bounds for that node. L and U are the global lower and upper bounds. A superscript is used if the node was repeated due to significant bounds tightening during postprocessing.

In this process, preprocessing included feasibility-based range reduction (poor man's LPs) and range reduction based on an objective function cut. In the postprocessing step, bounds were tightened using the marginal values obtained during lower bounding. Probing was employed for variables which were strictly within their bounds. The relaxed problem was constructed by underestimating each concave quadratic term in the objective by a linear term, within the variable bounds.

At the root node, preprocessing is performed by minimizing and maximizing each variable subject to the problem constraints. Any feasible solution obtained was used to update the upper bound. Even though this step did not yield improved variable bounds, an upper bound of -14 was obtained for the optimal objective.

Iteration 1: The lower bounding problem is solved at $x = (0.3, 1, 1, 1, 1)$, with $L_1 = -18.9$. No bounds tightening was possible by postprocessing. The node is branched at $x_1 = 0.3$. At this time, the global bounds are $L = -18.9$ and $U = -14.0$.

Iteration 2: Here, the node with $(0, 0, 0, 0, 0) \leq x \leq (0.3, 1, 1, 1, 1)$ is considered. During preprocessing, the variable domains contract to $(0, 0.953, 0.952, 0.950, 0) \leq x \leq (0.064, 1, 1, 1, 1)$. The relaxed problem yields $L_2 = -16.5$, with $x = (0, 1, 1, 1, 1)$. This value of x is also feasible to the original problem with $U_2 = -16.5$. As the upper and lower bounds in this node have collapsed, this node is fathomed. After this iteration, the global bounds are $L = -18.9$ and $U = -16.5$.

Iteration 3: Now the node with $(0.3, 0, 0, 0, 0) \leq x \leq (1, 1, 1, 1, 1)$ is

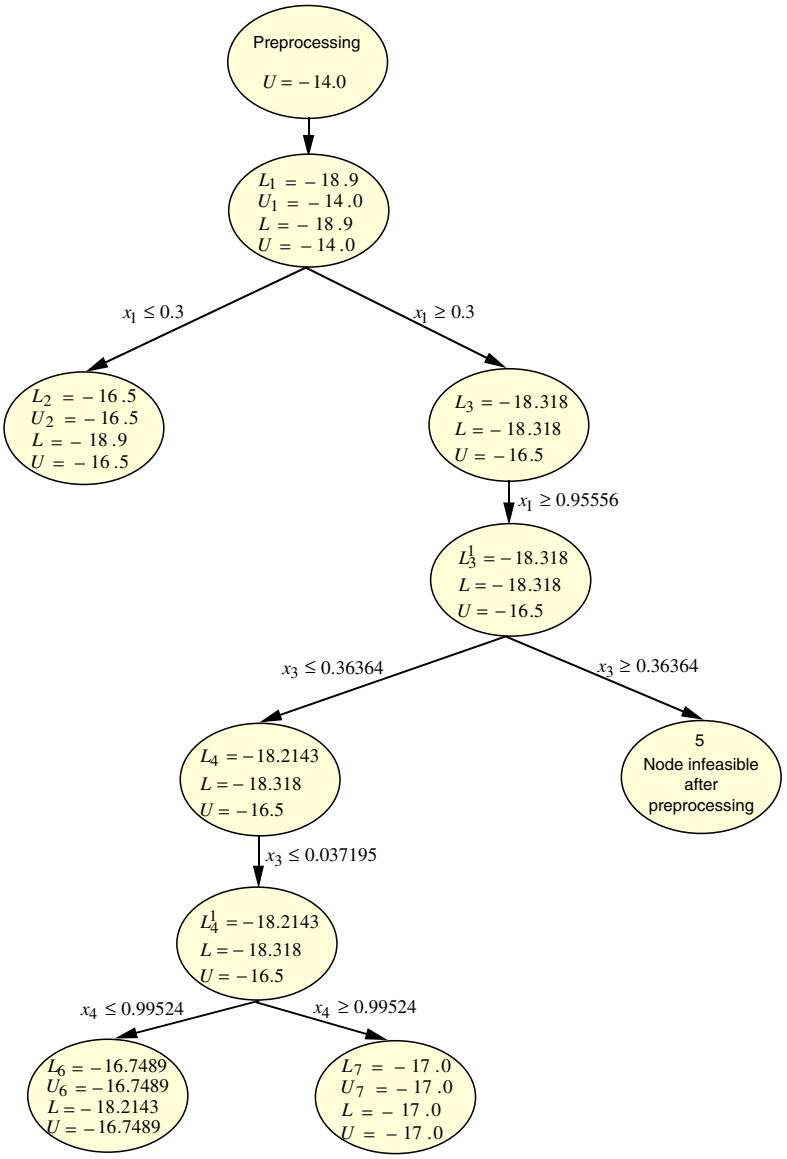

Figure 2.13: Branch and reduce tree for Example 2.

solved. In preprocessing, the bounds are tightened to $(0.84, 0, 0, 0, 0) \leq x \leq (1, 1, 1, 1, 1)$. The relaxed problem is solved with $x = (1, 1, 0.364, 0, 1)$, and $L_3 = -18.318$. Now the dual multiplier corresponding to the variable bound constraint $x_1 - 1 \leq 0$ is $\lambda_1 = 40.909$. This gives the bound $x_1 \geq x_1^u - (U - L_3)/\lambda_1$, yielding $0.956 \leq x_1 \leq 1$. As the bounds on x_1 have been tightened significantly, this node is solved again with the updated bounds. This again yields the solution $x = (1, 1, 0.364, 0, 1)$ and $L_3^1 = -18.318$. Tightening using the marginals during postprocessing yields $0.961 \leq x_1 \leq 1$. At this stage, the node is branched at $x_3 = 0.364$. After this iteration, the bounds are $L = -18.318$ and $U = -16.5$.

Iteration 4: The node with $(0.961, 0, 0, 0, 0) \leq x \leq (1, 1, 0.364, 1, 1)$ is solved here. After preprocessing, the variable bounds are $(0.961, 0.944, 0, 0, 0) \leq x \leq (1, 1, 0.0725, 1, 1)$. The relaxed problem yields $x = (1, 1, 0, 0.571, 1)$ and $L_4 = -18.214$. Postprocessing using marginals and probing contracts the bounds to $(0.964, 0.964, 0, 0, 0) \leq x \leq (1, 1, 0.0372, 1, 1)$. The node is solved again with the new bounds. The solution obtained is $x = (1, 1, 0, 0.571, 1)$ and $L_4^1 = -18.214$. Postprocessing on the node yields $(0.991, 0.991, 0, 0.99, 0) \leq x \leq (1, 1, 0.0112, 1, 0.0106)$. The current node is branched at $x_4 = 0.99524$. After this iteration, the bounds are $L = -18.318$ and $U = -16.5$.

Iteration 5: The node with $(0.961, 0, 0.364, 0, 0) \leq x \leq (1, 1, 1, 1, 1)$ is solved here. During preprocessing, this node becomes infeasible, indicating that all points in it are "inferior" to the current best upper bound. Hence, this node is fathomed. At this stage, the bounds are $L = -18.214$ and $U = -16.5$.

Iteration 6: The node with $(0.991, 0.991, 0, 0.99, 0) \leq x \leq (1, 1, 0.0112, 0.995, 0.0106)$ is solved here. After preprocessing, the bounds become $(0.996, 0.996, 0, 0.99, 0) \leq x \leq (1, 1, 0.00556, 0.995, 0.00527)$. The solution of the relaxed problem is $x = (1, 1, 0, 0.995, 0)$ with $L_6 = -16.7489$. This value of x is also feasible to the original problem with $U_6 = -16.749$. Since the lower and upper bounds are same for this node, it is fathomed. After this iteration, $L = -18.214$ and $U = -16.74$.

Iteration 7: The node with $(0.991, 0.991, 0, 0.995, 0) \leq x \leq (1, 1, 0.0112, 1, 0.0106)$ is solved now. After preprocessing, the variable bounds become $(0.996,\ 0.996, 0, 0.995, 0) \leq x \leq (1, 1, 0.00562, 1, 0.00532)$. The minimum for the relaxed problem occurs at $x = (1, 1, 0, 1, 0)$ with $L_7 = -17.0$. This point is also feasible to the original quadratic problem with $U_7 = -17.0$. As the bounds for this node have collapsed, this node is fathomed. After this iteration, $L = -17.0$ and $U = -17.0$.

As there are no more open nodes, the algorithm terminates at this point with $x = (1, 1, 0, 1, 0)$ and $f = -17.0$.

As can be seen, range reduction and other acceleration techniques greatly enhance the effectiveness of the branch and bound algorithm.

Chapter 3

System Features

3.1 Core Component

BARON comes in the form of a callable library. The software has a *core* component which can be used to solve **any** global optimization problem for which the user supplies problem specific subroutines, primarily for lower and upper bounding. In this way, the core system is capable of solving very general problems. The usage of the core component is detailed in Chapter 7.

3.2 Specialized Modules

In addition to the general purpose core, the BARON library also provides ready to use *specialized modules* covering several classes of problems. These modules work in conjunction with the core component and require no coding on the part of the user as detailed in Chapter 8. In this chapter, we restrict the discussion to a description of the capabilities of the modules.

> **Note**: In all modules, x_i is integer for $i = 1, \ldots, r$, so that decision and integer variables may be used.

3.2.1 Separable Concave Quadratic Programming

$$(SCQP): \qquad \min \quad f(x) = \sum_{i=1}^{n} (c_i x_i + q_i x_i^2)$$

$$\begin{aligned} \text{s.t.} \quad & l_r \leq Ax \leq u_r \\ & l_c \leq x \leq u_c \\ & x_i \in \mathbb{Z} \text{ for } i = 1, \ldots, r \\ & x_i \in \mathbb{R} \text{ for } i = r+1, \ldots, n \end{aligned}$$

where $x \in \mathbb{Z}^r \times \mathbb{R}^{n-r}$, $c \in \mathbb{R}^n$, $q \in \mathbb{R}^n_-$, $l_r \in \mathbb{R}^m$, $u_r \in \mathbb{R}^m$, $l_c \in \mathbb{R}^n$, $u_c \in \mathbb{R}^n$, and $A \in \mathbb{R}^{m \times n}$.

3.2.2 Separable Concave Programming

$$\begin{aligned} (SCP): \quad \min \quad & f(x) = \sum_{i=1}^{n} f_i(x_i) \\ \text{s.t.} \quad & l_r \leq Ax \leq u_r \\ & l_c \leq x \leq u_c \\ & x_i \in \mathbb{Z} \text{ for } i = 1, \ldots, r \\ & x_i \in \mathbb{R} \text{ for } i = r+1, \ldots, n \end{aligned}$$

where $x \in \mathbb{Z}^r \times \mathbb{R}^{n-r}$, $f_i(x_i)$ are concave and bounded over $l_{c,i} \leq x_i \leq u_{c,i}$ $(i = 1, \ldots, n)$, $l_r \in \mathbb{R}^m$, $u_r \in \mathbb{R}^m$, $l_c \in \mathbb{R}^n$, $u_c \in \mathbb{R}^n$, and $A \in \mathbb{R}^{m \times n}$.

3.2.3 Problems with Power Economies of Scale (Cobb-Douglas functions)

$$\begin{aligned} (PES): \quad \min \quad & f(x) = \sum_{i=1}^{n} c_i x_i^{q_i} \\ \text{s.t.} \quad & l_r \leq Ax \leq u_r \\ & l_c \leq x \leq u_c \\ & x_i \in \mathbb{Z} \text{ for } i = 1, \ldots, r \\ & x_i \in \mathbb{R} \text{ for } i = r+1, \ldots, n \end{aligned}$$

where $x \in \mathbb{Z}^r \times \mathbb{R}^{n-r}$, $c \in \mathbb{R}^n_+$, $q \in (0,1]^n$, $l_r \in \mathbb{R}^m$, $u_r \in \mathbb{R}^m$, $l_c \in \mathbb{R}^n_+$, $u_c \in \mathbb{R}^n_+$, and $A \in \mathbb{R}^{m \times n}$.

3.2.4 Fixed-Charge Programming

$$(FCP): \quad \min \quad f(x) = \sum_{i=1}^{n} f_i(x_i)$$

$$\begin{aligned} \text{s.t.} \quad & l_r \leq Ax \leq u_r \\ & l_c \leq x \leq u_c \\ & x_i \in \mathbb{Z} \text{ for } i = 1, \ldots, r \\ & x_i \in \mathbb{R} \text{ for } i = r+1, \ldots, n \end{aligned}$$

where $x \in \mathbb{Z}^r \times \mathbb{R}^{n-r}$, $f_i(x_i)$ equals $c_i + q_i x_i$ if $x_i > 0$ and 0 otherwise $(i = 1, \ldots, n)$, $c \in \mathbb{R}^n_+$, $q \in \mathbb{R}^n$, $l_r \in \mathbb{R}^m$, $u_r \in \mathbb{R}^m$, $l_c \in \mathbb{R}^n_+$, $u_c \in \mathbb{R}^n_+$, and $A \in \mathbb{R}^{m \times n}$.

3.2.5 Fractional Programming

$$\begin{aligned} (FP): \qquad \min \quad & f(x) = \frac{cx + \alpha}{qx + \beta} \\ \text{s.t.} \quad & l_r \leq Ax \leq u_r \\ & l_c \leq x \leq u_c \\ & x_i \in \mathbb{Z} \text{ for } i = 1, \ldots, r \\ & x_i \in \mathbb{R} \text{ for } i = r+1, \ldots, n \end{aligned}$$

where $x \in \mathbb{Z}^r \times \mathbb{R}^{n-r}$, $c \in \mathbb{R}^n$, $q \in \mathbb{R}^n$, $\alpha \in \mathbb{R}$, $\beta \in \mathbb{R}$, $l_r \in \mathbb{R}^m$, $u_r \in \mathbb{R}^m$, $l_c \in \mathbb{R}^n$, $u_c \in \mathbb{R}^n$, and $A \in \mathbb{R}^{m \times n}$. We require $qx + \beta > 0$ for any $x \in X := \{x \mid l_r \leq Ax \leq u_r \text{ and } l_c \leq x \leq u_c\}$, and X compact. It is well known that this problem can be transformed to an equivalent linear program if no integer variables are present.

3.2.6 Univariate Polynomial Programming

$$\begin{aligned} (POLY): \qquad \min \quad & f(x) = \sum_{i=0}^{k} c_i x^i \\ \text{s.t.} \quad & l_c \leq x \leq u_c \end{aligned}$$

where $x \in \mathbb{R}$, $c \in \mathbb{R}^k$, $l_c \in \mathbb{R}$, $u_c \in \mathbb{R}$.

3.2.7 Linear Multiplicative Programming

$$\begin{aligned} (LMP): \qquad \min \quad & f(x) = \prod_{i=1}^{p} f_i(x) = \prod_{i=1}^{p} \left(c_i^t x + c_{i0}\right) \\ \text{s.t.} \quad & l_r \leq Ax \leq u_r \end{aligned}$$

$$
\begin{aligned}
& l_c \leq x \leq u_c \\
& x_i \in \mathbb{Z} \text{ for } i = 1, \ldots, r \\
& x_i \in \mathbb{R} \text{ for } i = r+1, \ldots, n
\end{aligned}
$$

where $x \in \mathbb{Z}^r \times \mathbb{R}^{n-r}$, $c_i \in \mathbb{R}^n$ and $c_{i0} \in \mathbb{R}$ $(i = 1, \ldots, p)$, $l_r \in \mathbb{R}^m$, $u_r \in \mathbb{R}^m$, $l_c \in \mathbb{R}^n$, $u_c \in \mathbb{R}^n$, and $A \in \mathbb{R}^{m \times n}$. We require $c_i^t x + c_{i0} \geq 0$ $(i = 1, \ldots, p)$ for $x \in \{x \mid l_r \leq Ax \leq u_r \text{ and } l_c \leq x \leq u_c\}$.

3.2.8 General Linear Multiplicative Programming

$$
\begin{aligned}
(GLMP): \quad \min \quad & f(x) = \sum_{i=1}^{T} \prod_{j=1}^{p_i} \left(c_{ij}^0 + c_{ij}^t x\right) \\
\text{s.t.} \quad & l_r \leq Ax \leq u_r \\
& l_c \leq x \leq u_c \\
& x_i \in \mathbb{Z} \text{ for } i = 1, \ldots, r \\
& x_i \in \mathbb{R} \text{ for } i = r+1, \ldots, n
\end{aligned}
$$

where $x \in \mathbb{Z}^r \times \mathbb{R}^{n-r}$, $c_{ij} \in \mathbb{R}^n$ and $c_{ij}^0 \in \mathbb{R}$ $(i = 1, \ldots, T; j = 1, \ldots, p_i)$, $l_r \in \mathbb{R}^m$, $u_r \in \mathbb{R}^m$, $l_c \in \mathbb{R}^n$, $u_c \in \mathbb{R}^n$, and $A \in \mathbb{R}^{m \times n}$.

3.2.9 Indefinite Quadratic Programming

$$
\begin{aligned}
(IQP): \quad \min \quad & f(x) = \sum_{i=1}^{n} \sum_{j=1}^{n} q_{ij} x_i x_j + \sum_{i=1}^{n} c_i x_i \\
\text{s.t.} \quad & l_r \leq Ax \leq u_r \\
& l_c \leq x \leq u_c \\
& x_i \in \mathbb{Z} \text{ for } i = 1, \ldots, r \\
& x_i \in \mathbb{R} \text{ for } i = r+1, \ldots, n
\end{aligned}
$$

where $x \in \mathbb{Z}^r \times \mathbb{R}^{n-r}$, $q \in \mathbb{R}^{n \times n}$, $c \in \mathbb{R}^n$, $l_r \in \mathbb{R}^m$, $u_r \in \mathbb{R}^m$, $l_c \in \mathbb{R}^n$, $u_c \in \mathbb{R}^n$, and $A \in \mathbb{R}^{m \times n}$.

3.2.10 Mixed Integer Linear Programming

$$
(MILP): \quad \min \quad f(x) = \sum_{i=1}^{n} c_i x_i
$$

$$\begin{aligned} \text{s.t.} \quad & l_r \leq Ax \leq u_r \\ & l_c \leq x \leq u_c \\ & x_i \in \mathbb{Z} \text{ for } i = 1, \ldots, r \\ & x_i \in \mathbb{R} \text{ for } i = r+1, \ldots, n \end{aligned}$$

where $x \in \mathbb{Z}^r \times \mathbb{R}^{n-r}$, $c \in \mathbb{R}^n$, $l_r \in \mathbb{R}^m$, $u_r \in \mathbb{R}^m$, $l_c \in \mathbb{R}^n$, $u_c \in \mathbb{R}^n$, and $A \in \mathbb{R}^{m \times n}$.

3.2.11 Mixed Integer Semidefinite Programming

$$\begin{aligned} (MISDP): \quad \min \quad & f(x) = \sum_{i=1}^{n} c_i x_i \\ \text{s.t.} \quad & \sum_{i=1}^{n} F_i x_i - F_0 \text{ is positive semidefinite } (\succeq 0) \\ & l_c \leq x \leq u_c \\ & x_i \in \mathbb{Z} \text{ for } i = 1, \ldots, r \\ & x_i \in \mathbb{R} \text{ for } i = r+1, \ldots, n \end{aligned}$$

where $x \in \mathbb{Z}^r \times \mathbb{R}^{n-r}$, $c \in \mathbb{R}^n$, $l_c \in \mathbb{R}^n$, $u_c \in \mathbb{R}^n$, and $F_i \in \mathbb{R}^{m \times m}$ $(i = 0, \ldots, n)$ are symmetric matrices.

3.3 Factorable Nonlinear Programming Module

This is the most general of the supplied modules that do not require any coding from the user. The input to this module is through the parser, which is described in Chapter 9.

The module solves fairly general nonlinear programs:

$$\begin{aligned} (NLP): \quad \min \quad & f(x) \\ \text{s.t.} \quad & l_r \leq g(x) \leq u_r \\ & l_c \leq x \leq u_c \\ & x_i \in \mathbb{Z} \text{ for } i = 1, \ldots, r \\ & x_i \in \mathbb{R} \text{ for } i = r+1, \ldots, n \end{aligned}$$

where, $x \in \mathbb{Z}^r \times \mathbb{R}^{n-r}$, $f : \mathbb{R}^n \to \mathbb{R}$, $g : \mathbb{R}^n \to \mathbb{R}^m$, $l_r \in \mathbb{R}^m$, $u_r \in \mathbb{R}^m$, $l_c \in \mathbb{R}^n$, $u_c \in \mathbb{R}^n$, and f, g are factorable functions.

Factorable functions are recursive compositions of sums and products of functions of single variables. Most functions of several variables used in nonlinear optimization are factorable and can be easily brought into separable form [14].

Currently, the types of functions allowed in this module include $\exp(x)$, $\ln(x)$, x^α for $\alpha \in \mathbb{R}$, and β^x for $\beta \in \mathbb{R}$. In this way, the capabilities of this module subsume those of all other specialized modules. However, this module is not designed to fully exploit the special structure of the problems addressed by the above-described specialized modules.

Chapter 4

Hardware and Software Requirements

The code is currently available for UNIX systems and can be ported to other platforms as well. Most specialized modules require the use of a linear programming solver as shown in Table 4.1. An NLP (QP) solver is optional and frequently useful in the NLP (IQP) module.

Feature	LP solver	SDP solver
Core BARON		
NLP	√	
SCQP	√	
SCP	√	
PES	√	
FCP	√	
FP	√	
POLY		
LMP	√	
GLMP	√	
IQP	√	
MILP	√	
MISDP		√

Table 4.1: Solver requirements of BARON modules

Currently, we provide interfaces to OSL [9], CPLEX [3], MINOS [15], SDPA [6], and SNOPT [7] for solving the required linear, nonlinear, or

semidefinite programming subproblems. The solver options are as follows:

- The LP solver can be either CPLEX, OSL, MINOS, or SNOPT in all modules except for `FP` which runs only under OSL.
- The desired solver is specified in the `options` file as described in Chapter 6.
- General NLP solvers for which we currently provide an interface are MINOS and SNOPT.
- Instead of an LP solver, the `IQP` module can optionally use the QP solver of OSL, MINOS or SNOPT.
- The only SDP solver currently supported by the `MISDP` module is SDPA.

Future plans include the addition of interfaces to other solvers as well.

Chapter 5

Installation

The code can be downloaded from `http://archimedes.scs.uiuc.edu`.

5.1 Core Component and Specialized Modules

BARON comes in the form of a library called `libbaron.a`. A sample FORTRAN file `main.f`, which calls BARON is also supplied along with a makefile to link to all required subroutine libraries such as BLAS, CPLEX, LAPACK, MINOS, OSL, SDPA, or SNOPT. In the simplest possible form, the `main.f` file can be compiled using the command:

```
xlf -c main.f -O
xlf main.o -lbaron -O -o main
```

Note that the order in which arguments appear in the `xlf` command is important so as to overwrite the default BARON subroutines when the user provides specialized routines.

If the user has access to a library `mylib` (e.g., OSL, CPLEX, SNOPT, or MINOS) that is required (most specialized modules require one LP solver), the following command sequence will be required:

```
xlf -c main.f -O
xlf main.o -mylib -lbaron -O -o main
```

In both the above cases, the executable is put in the file `main`.

5.2 Parser

The parser part of BARON comes in the file `libbarin.a`. This library has to be linked to the main file `barin.c` to produce an executable. For example, if the final executable is to be called `barin`, then the compilation can be done as follows:

```
cc -c barin.c
xlf barin.o -lbaron -lbarin -o barin
```

If the user has access to a library `mylib` (*e.g.*, BLAS, CPLEX, LAPACK, MINOS, OSL, SDPA, or SNOPT) that is required (most specialized modules require one LP solver), the following command sequence will be required:

```
cc -c barin.c
xlf barin.o -mylib -lbaron -lbarin -o barin
```

Chapter 6

Algorithmic and System Options

Before describing how to use the core BARON or specialized modules, in this chapter we detail the algorithmic and other options that are available to the user. All options come with a default value and it is, therefore, not necessary for the user to modify any options. The ability to modify them, however, provides a great deal of flexibility.

6.1 Termination Options

Option	Description	Default Value
`epsa` (ϵ_a)	Absolute convergence tolerance	10^{-6}
`epsr` (ϵ_r)	Relative convergence tolerance	0
`maxtime`	Maximum CPU time allowed (sec)	1200
`maxiter`	Maximum number of branch and bound iterations allowed	-1

The algorithm terminates if any of the following conditions is satisfied:

- the CPU time limit, `maxtime`, has been reached or exceeded;
- the iterations limit, `maxiter`, has been reached;
- $U - L \leq \epsilon_a$;
- $U - L \leq \epsilon_r |L|$.

where L and U are the lower and upper bound on the global minimum at a given iteration. Setting `maxiter` equal to -1 places no limit on the number of iterations.

6.2 Branching Options

Option	Description	Default Value
`brstra`	Branching strategy	0
`modbrpt`	Branch point modification option	1
`numbranch`	Number of variables to be branched on	-1
`numstore`	Number of variables whose bounds are to be stored at every node of the tree	0

$$\texttt{brstra} = \begin{cases} 0, & \text{if omega branching is desired,} \\ 1, & \text{if bisection is desired.} \end{cases}$$

$$\texttt{modbrpt} = \begin{cases} 0, & \text{user-specified branching point is used without any modifications,} \\ 1, & \text{allows BARON to modify the user-specified branching point, if applicable.} \end{cases}$$

$$\texttt{numbranch} = \begin{cases} -1, & \text{consider all variables for branching,} \\ n, & \text{consider only the first } n \text{ variables for branching.} \end{cases}$$

$$\texttt{numstore} = \begin{cases} 0, & \text{store } \texttt{numbranch} \text{ variables,} \\ -1, & \text{store all variables,} \\ n, & \text{store } n \text{ variables.} \end{cases}$$

Only the first `numbranch` variables are branched on. By default, all variables are considered for branching and the bounds of all branching variables are stored. Except for the default case (`numstore` $= 0$, implying `numstore` $=$ `numbranch`), `numbranch` $\leq$ `numstore` is required.

6.3 Heuristic Local Search Options

Option	Description	Default Value
`dolocal`	Local search option for upper bounding	1
`maxheur`	Maximum number of passes allowed for heuristic	5
`habstol`	Absolute improvement in the objective to repeat heuristic	0.1
`hreltol`	Relative improvement in the objective to repeat heuristic	0.1
`numloc`	Number of local searches done in NLP preprocessing	1
`locres`	Option to control output from local search	0

$$\texttt{dolocal} = \begin{cases} 0, & \text{no local search is done during upper bounding,} \\ 1, & \text{local search is used for upper bounding,} \\ -n, & \text{local search is done once every } n \text{ iterations.} \end{cases}$$

At a given node, a local search heuristic is performed by calling subroutine `user6` (discussed in Chapter 7) if the value of `dolocal` so dictates. The call to `user6` is repeated after heuristic starting point initialization up to `maxheur` times as long as the upper bound improvement during two consecutive passes satisfies either the relative or absolute improvement requirement. Here, `habstol` and `hreltol` can take any nonnegative value.

In the preprocessing step of the NLP module, `numloc` local searches are done. The first one begins with the user-specified starting point as long as it is feasible. Subsequent local searches are done from randomly generated starting points. If `locres` is set to 1, detailed results from each local search will be printed to the results file and a listing of all the different objective function values obtained at the end of each local search will appear on the screen along with a summary of the number of occurrences of each different objective function value obtained.

6.4 Range Reduction Options

Option	Description	Default Value
`tdo`	Bounds tightening option	1
`mdo`	Marginals testing option	1
`lbttdo`	Poor man's LP option	1
`obttdo`	Optimality-based tightening option	1
`pdo`	Number of probing problems allowed	0
`pxdo`	Number of probing problems with an x-objective (`pxdo` $\leq$ `pdo` (full probing)	0
`profra`	Fraction of probe to bound distance from relaxed solution when forced probing is done	0.67
`twoways`	Determines whether probing on both bounds is done or not	1
`maxredpass`	Maximum number of times range reduction is performed at a node before a new relaxation is constructed	10
`maxnodepass`	Maximum number of passes (relaxation constructions) allowed through a node	5
`creltol`	Relative improvement in the objective to reconstruct the relaxation of the current node	0.1
`cabstol`	Absolute improvement in the objective to reconstruct the relaxation of the current node	0.1

$$\texttt{tdo} = \begin{cases} 0, & \text{if no bounds tightening is to be performed,} \\ 1, & \text{if bounds tightening is to be performed.} \end{cases}$$

$$\texttt{mdo} = \begin{cases} 0, & \text{if range reduction based on marginals is not to be done,} \\ 1, & \text{if range reduction based on marginals is to be done.} \end{cases}$$

$$\texttt{pdo} = \begin{cases} 0, & \text{if range reduction by probing is not desired,} \\ -1, & \text{if probing on all variables is desired,} \\ n, & \text{if probing on } n \text{ variables is desired.} \end{cases}$$

$$\texttt{twoways} = \begin{cases} 0, & \text{if probing is to be done at the farthest bound,} \\ 1, & \text{if probing is to be done at both bounds.} \end{cases}$$

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. Only feasibility-based tightening is done during preprocessing. If postprocessing improves the node's lower bound in a way that satisfies the absolute or relative tolerances, `cabstol` or `creltol`, respectively, the process of lower bounding followed by postprocessing is repeated up to `maxnodepass` times. Here, `cabstol` and `creltol` can take any nonnegative value.

6.5 Module Options

The option `alg` can take integer values corresponding to different algorithms. The possible values for `alg` are given below:

`alg`	Meaning
0	Core component without any specialized module
1	Separable Concave Quadratic Programming (SCQP)
2	Separable Concave Programming (SCP)
3	Power Economies of Scale (PES)
4	Fixed Charge Programming (FCP)
5	Fractional Programming (FP)
6	Univariate Polynomial Programming (POLY)
7	Linear Multiplicative Programming (LMP)
8	General Linear Multiplicative Programming (GLMP)
9	Indefinite Quadratic Programming (IQP)
10	Mixed Integer Linear Programming (MILP)
11	Mixed Integer Semidefinite Programming (MISDP)
99	Local Search from a number of random points
100	Factorable Nonlinear Programming (NLP)

6.6 Output Options

Option	Description	Default Value
`prfreq`	Results are printed every `prfreq` nodes	100
`prlevel`	Level of results printed	1
`results`	If equal to 1, a results file `res.lst` will be created	1
`summary`	If equal to 1, a summary file `sum.lst` will be created	1
`times`	If equal to 1, a file `tim.lst` with a breakdown of CPU time spent on sections of BARON will be provided	1

Every `prfreq` nodes, the current lower and upper bounds are printed on the screen and the summary file. All output is suppressed if `prlevel` is nonpositive. Larger values of `prlevel` produce more output. The `sum.lst` contains a copy of the bounds printed on the screen whereas `res.lst` contains all successively improved solutions found during the course of the algorithm. The results, summary or time files will not be created if the corresponding option is set to 0.

6.7 Other Options

Option	Description	Default Value
`lpsol`	LP solver to be used	3
`qpsol`	QP solver to be used	2
`nlpsol`	NLP solver to be used	1
`sdpsol`	SDP solver to be used	6
`lpsolopt`	Read LP solver options file if 1	0
`qpsolopt`	Read QP solver options file if 1	0
`nlpsolopt`	Read NLP solver options file if 1	0
`sdpsolopt`	Read SDP solver options file if 1	0
`nlpdolin`	Linearization option for NLP module	1
`qplin`	Linearization option for IQP module	0
`numint`	Number of integer variables	0
`usave`	Length of user array at each node	0
`baskp`	Indicates whether basis information is to be saved	0
`baslen`	Length of basis array	0
`basfra`	Similarity measure between bases for basis update not to occur	0.7
`usave`	Length of user array at each node	0
`postabstol`	Absolute tolerance for postponing a node	10^{30}
`postreltol`	Relative tolerance for postponing a node	10^{30}
`prelpdo`	Solve preprocessing LPs at root if 1	1
`presdpdo`	Solve preprocessing SDPs at root if 1	1
`diagonalize`	Chooses among a separable and a non-separable formulation in IQP	1
`cutoff`	Eliminate solutions that are no better than this value	`infinity`

`lpsol` and `qpsol` are used to specify the choice of LP and QP solver, respectively. Possible values are 1 for OSL, 2 for MINOS, 3 for CPLEX, and 4 for SNOPT. The NLP solver is specified using `nlpsol`. A value of 2 corresponds to MINOS and a value of 4 selects SNOPT. The SDP solver is specified using `sdpsol`. A value of 6 corresponds to SDPA.

A solver options file will be read if the corresponding option `lpsolopt`, `qpsolopt`, `nlpsolopt`, or `sdpsolopt` is set to 1. The options file has a name

of the form `minoslp.opt`, `minosqp.opt`, `minosnlp.opt`, or `sdpasdp.opt`.

The `nlpdolin` option applies only to the NLP module. A value of 1 will result in the use of a linear programming relaxation whereas a value of 0 will result in the use of nonlinear relaxations whenever possible. A linear relaxation is used by default.

The `qplin` option is relevant to the IQP module only. If `qplin` is set to 1, a linear relaxation is used by these modules; a quadratic relaxation is used by default.

The first `numint` variables of the problem will be treated as integers for branching and tightening purposes.

The array `userdata`, which is of length `usave` *for each node* of the tree, is available for storing node-specific information. This array can be accessed in most of the user exits as described in Chapter 7. When a node is partitioned, its `userdata` array is inherited by its two descendants.

The basis array `basis` is of length `baslen` *for each node* of the tree and is passed to `user2` which is also expected to update the array upon exit. The option `baskp` may take different nonzero values for different basis information saving schemes. The default value of 0 corresponds to saving no basis information. Whenever `baskp` is nonzero, the modules' 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.

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 z_2 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:

- $z - z_2 \geq$ `postabstol`
- $z - z_2 \geq$ `postreltol` $\times |z_2|$.

The `prelpdo` option applies to all specialized modules while the `presdpdo` option applies to the SDP module only. The `diagonalize` option applies to the IQP module only.

6.8 The `options` File

All options are initialized in BARON at their default values. If the user wishes to modify any options, this must be communicated to BARON through the `options` file. This file contains one line per option that needs

to be modified with the option name followed by its value. Options that are not named in this file take their default values. If an `options` file is not present in the directory where BARON runs, all options will assume their default value. A typical `options` file is shown below:

```
*This is a typical BARON options file.
*We will rely on the default BARON options with two exceptions.
brstra  1            *Branching strategy.
alg     9            *We will use the specialized module for MIQP.
times   1
*prfreq 100
*maxnodepass 20
*maxredpass 10
*maxheur 10
```

In this file, a line or part of line following a '*', '!' or '#' will be treated as a comment and ignored.

Note: The number of integer variables is passed through the `options` file described only when the core component of BARON is used. For the specialized modules, `numint` is not specified in the `options` file. It is either passed directly through the corresponding subroutine call or included in the `problem` file as described in Chapter 8.

Chapter 7

Using The Core Component

The core component of BARON can be programmed to solve very general types of global optimization problems. A main file must be supplied that calls subroutine `baron`. Seven additional subroutines, `user1`–`user7`, are required and will be called by `baron`. The BARON subroutines, data structures and heuristics, in conjunction with the user written subroutines, solve the problem to global optimality (Figure 7.1). A description of the required subroutines is given in this chapter.

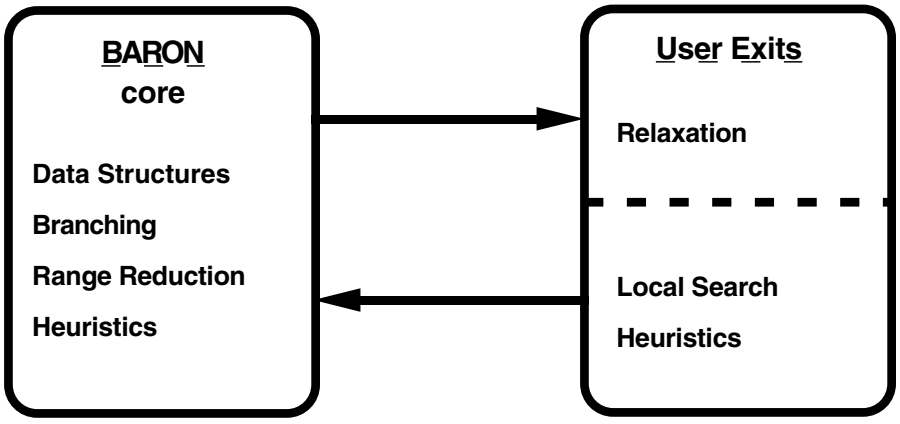

Figure 7.1: Core-user interaction.

7.1 Subroutine `baron`

Purpose:	Global optimization using BARON.	
Usage:	`call baron`	`(lc, uc, n, barspace, work, optfile, status, lbest, ubest, xbest, totaltime)`
	`integer*4`	`n, status, work`
	`parameter`	`(n=<value>, work=<value>)`
	`real*8`	`lc(n), uc(n), xbest(n), barspace(work), lbest, ubest, totaltime`
	`character*7`	`optfile`
Arguments:		
Inputs:	`lc, uc`	lower and upper bounds on problem variables.
	`n`	number of variables in the problem.
	`barspace`	work array for BARON: all memory-intensive information is stored here.
	`work`	length of barspace. The number of doublewords needed is approximately $\texttt{maxsub} \times (2\texttt{n} * \texttt{usave} + 2.5) + 8\texttt{n} + 100$, where `maxsub` is the maximum number of subproblems to be allowed in memory at any point during the search. Based on `work`, BARON will calculate the allowed `maxsub`.
	`optfile`	name of the options file.
Outputs:	`status`	1, if successful termination; 2, if max. no. of nodes allowed exceeded; 3, if max. no. of iterations allowed exceeded; 4, if max. CPU time allowed exceeded.
	`xbest`	the solution vector corresponding to the best upper bound at termination.
	`lbest`	the best lower bound on the problem objective function.
	`ubest`	the best upper bound on the problem objective function.
	`totaltime`	total time (CPU seconds) taken for the global minimization process.

Note that the array `xbest` may contain the best known solution at entry in which case BARON will use this as an upper bound.

7.2 User Subroutines

The following problem-specific subroutines must be provided by the user. Even though subroutines `user1`, `user4` and `user6` are optional, providing them is often essential for improved performance.

7.2.1 Subroutine `user1` (Range Reduction)

Purpose:	Variable bounds tightening using optimality-based and feasibility-based range-reduction tests. This subroutine is called before and after lower/upper bounding a node.	
Usage:	`subroutine user1`	`(lc, uc, ubest, userdata, n, success)`
	`integer*4`	`success, n`
	`real*8`	`lc(n), uc(n), ubest, userdata(usave)`
Arguments:		
Inputs:	`lc, uc`	variable bounds for current node.
	`ubest`	current best upper bound on the objective function.
	`userdata`	userdata for current node (initialized at -99999 for the root node).
Outputs:	`lc, uc`	tightened lower and upper bounds.
	`userdata`	userdata for current node.
	`success`	1, if successful tightening; 0, otherwise.

Remarks:

- In addition to the original problem constraints, the relationships introduced for convexification and objective function cuts can be used for range reduction.
- Unless `numint` has been passed to BARON, integrality requirements must be enforced in this subroutine by rounding up lower bounds and

rounding down upper bounds of the integer variables.

7.2.2 Subroutine `user2` (Lower Bounding)

Purpose:	Lower bounding for current node.	
Usage:	`subroutine user2`	`(lc, uc, y, x, basis, userdata, robj, zl2, zub, marg, n, inform, ptype, pvar, pbnd, pobj)`
	`integer*4`	`n, inform, ptype, pvar, y`
	`real*8`	`lc(n), uc(n), y(n), x(n), userdata(usave), basis(baslen), robj, marg(n), pbnd, pobj(n)`
Arguments:		
Inputs:	`lc, uc`	variable bounds for current node.
	`ptype`	determines probing type with the following possible values: 0, if `user2` is not called for probing; +1 (−1), if probing at upper (lower) bound of variable `pvar` at point `pbnd` is desired; 2, if probing by optimizing the linear objective `pobj` over the relaxation constraint set is desired.
	`userdata`	userdata of closest related node, *i.e.*, current node if solved before, its parent otherwise = −99999, if no information is available).
	`basis`	basis of closest related node.
	`zl2`	second best lower bound on the objective.
	`zub`	current best upper bound on the objective.
Outputs:	`y`	relaxed problem solution.
	`x`	candidate solution of nonconvex problem (optional).

`marg`	marginal values (reduced costs) for problem variables.
`robj`	relaxed problem objective corresponding to `x`.
`inform`	−1: normal termination, feasible, do not branch, instead, postpone this node; 0: feasible, continue with branching; 1: infeasible; other value: abnormal termination.
`userdata`	userdata for current node.
`basis`	basis for current node.

Remarks:

- Underestimators must be constructed using the bounds `lc/uc`.
- If `pindex` is nonzero, the corresponding variable must be fixed at the correct bound before the node is solved. Note that the relaxation must be constructed using `lc/uc` and not a fixed value.

7.2.3 Subroutine `user3` (Feasibility Tester)

Purpose:	Feasibility checker for candidate solutions.	
Usage:	`subroutine user3`	`(x, n, isfeas)`
	`integer*4`	`n, isfeas`
	`real*8`	`x(n)`
Arguments:		
Input:	`x`	candidate solution.
Output:	`isfeas`	1, is candidate solution is feasible; 0, otherwise.

The user must decide on the tolerance to be used for constraint satisfaction. A commonly used value is 10^{-5}. However, constraint scaling should be taken into account.

7.2.4 Subroutine `user4` (Random Search)

Purpose:	Generate a starting point within specified bounds.	
Usage:	`subroutine user4`	`(lc, uc, x, n)`
	`integer*4`	`n`
	`real*8`	`lc(n), uc(n), x(n)`
Arguments:		
Inputs:	`lc, uc`	variable lower/upper bounds.
Output:	`x`	generated point.

The user can use a truncated stochastic global optimization algorithm or some other heuristic to generate the best possible solution to the problem. Alternatively, a random point can be generated.

7.2.5 Subroutine `user5` (Branching)

Purpose:	Calculation of violations and branching variable selection.	
Usage:	`subroutine user5`	`(lc, uc, x, userdata, viola, brpoint, n, brvar)`
	`integer*4`	`n, brvar`
	`real*8`	`lc(n), uc(n), x(n), userdata(usave), viola(n), brpoint`
Arguments:		
Inputs:	`lc, uc`	variable bounds for current node.
	`x`	point at which violations are to be calculated.
	`userdata`	userdata for current node (= −99999 if not available).
Outputs:	`viola`	deviation of underestimators from nonconvex functions assigned to individual variables.
	`brvar`	branching variable index.
	`brpoint`	branching position of the branching variable.

7.2.6 Subroutine `user6` (Local Search)

Purpose:	Upper bounding. A solution with a value better than `target` is sought, if possible.	
Usage:	`subroutine user6`	`(lc, uc, x, target, userdata, n)`
	`integer*4`	`n`
	`real*8`	`lc(n), uc(n), x(n), target, userdata(usave)`
Arguments:		
Inputs:	`lc, uc`	lower and upper variable bounds.
	`userdata`	userdata for current node (= −99999 if not available).
	`target`	target objective function value.
Output:	`x`	solution found.

7.2.7 Subroutine `user7` (Objective Function Evaluation)

Purpose:	Computation of nonconvex objective function value at a specified point (not necessarily feasible).	
Usage:	`subroutine user7`	`(x, obj, n)`
	`integer*4`	`n`
	`real*8`	`x(n), obj`
Arguments:		
Input:	`x`	specified point.
Output:	`obj`	nonconvex objective function value at `x`.

Chapter 8

Using the Specialized Modules

8.1 Input data and problem parameters

Any of the following methods can be used to input data to the specialized modules of BARON:

- By reading data files (the `problem` file).
- Through transferring data in memory using *callable subroutines*.
- In a high level format that will be read by the BARON parser.

The first two methods require an `options` file and are described below. The high level input format is described in Chapter 9.

The format for the `problem` file for each specialized module is described below through examples. There are two major options here, depending upon the value of `type` which is to appear early enough in the `problem` file:

- If `type` = 0, then the constraint matrix is input in *dense form*, *i.e.*, with `m` rows, `n` columns and `m` × `n` elements, including all zeros.
- `type` = 1 indicates that the constraint matrix is in the *column major* form. A matrix in column major form is represented by three one-dimensional arrays. The array `a` contains the nonzero elements of the matrix stored by columns. The array `ia` contains the row indices of the corresponding elements of `a` and is of the same length as `a`. The array `ja` contains the column starts for each column and is of length

equal to the number of columns plus one. The last element of `ja` is set equal to the number of nonzeros plus one.

Data can also be passed directly in memory by calling the subroutine pertaining to the corresponding module. To illustrate these subroutine calls, we will use the following notation.

Argument	Description	data type
`n`	Number of variables in the problem	`integer*4`
`m`	Number of constraints	`integer*4`
`ne`	Number of nonzeros in the constraint matrix	`integer*4`
`numint`	Number of integer variables	`integer*4`
`lc`	Array of lower bounds on variables	`real*8`
`uc`	Array of upper bounds on variables	`real*8`
`a`	Vector of constraint matrix nonzeros (by columns)	`real*8`
`ia`	Row indices of `a`	`integer*4`
`ia`	Column starts on `a`	`integer*4`
`barspace`	BARON work array	`real*8`
`work`	Length of the `barspace` array	`integer*4`
`optfname`	Name of the file containing options	`character*7`
`barstatus`	Exit status of BARON	`integer*4`
`zlbest`	Best lower bound	`real*8`
`zub`	Upper bound on objective	`real*8`
`xbest`	Array of best known solution	`real*8`
`totaltime`	Execution time of BARON	`real*8`
`proname`	Name of the problem	`character*8`

The value of `work` should be large enough to store all branch and bound data structures. If it is not, BARON will issue a related message and terminate. The arrays `lc`, `uc` and `xbest` should be of length greater than or equal to the actual number of problem variables (n).

Module-specific input variables will be explained in the following sections by means of examples.

8.2 Module `barscqp`: Separable Concave Quadratic Programming

$$\min \quad -10.5x_1 - 0.5x_1^2 - 7.5x_2 - 0.5x_2^2 - 3.5x_3 - 0.5x_3^2$$

$$
\begin{aligned}
& -2.5x_4 - 0.5x_4^2 - 1.5x_5 - 0.5x_5^2 - 10x_6 \\
\text{s.t.} \quad & 6x_1 + 3x_2 + 3x_3 + 2x_4 + x_5 \leq 6.5 \\
& 10x_1 + 10x_3 + x_6 \leq 20 \\
& 0 \leq x_i \leq 1 \text{for } i = 1, \ldots, 6
\end{aligned}
$$

The above problem can be entered using `type` = 0, in the following format.

```
scqp                                              problem name
2       6       0                                 m, n, numint
0                                                 type
       0    0    0    0    0     0                lc
-1e31  6    3    3    2    1     0    6.5         lr, a, ur (row 1)
-1e31 10    0   10    0    0     1    20          lr, a, ur (row 2)
       1    1    1    1    1     1e31             uc
     -10.5 -7.5 -3.5 -2.5 -1.5 -10                c
     -.5   -.5  -.5  -.5  -.5    0                q
```

In case the constraint matrix is in the column major form (*i.e.*, `type` = 1), the `problem` file looks as shown below:

```
scqp                                              problem name
2       6       0                                 m, n, numint
1                                                 type
8                                                 ne
6   10    3    3   10    2    1    1              a(i),  i=1,ne
1    2    1    1    2    1    1    2              ia(i), i=1,ne
1    3    4    6    7    8    9                   ja(i), i=1,n+1
-1e31   -1e31                                     lr(i), i=1,m
  6.5      20                                     ur(i), i=1,m
       0    0    0    0    0     0                lc
       1    1    1    1    1     1e31             uc
     -10.5 -7.5 -3.5 -2.5 -1.5 -10                c
     -.5   -.5  -.5  -.5  -.5    0                q
```

For either value of `type`, a starting point may be optionally supplied by providing an array of length `n` in the last row of the input file.

Problem data can also be passed directly in memory by calling subroutine `barscqp` as shown below. The arrays should first be declared as follows:

The `real*8` arrays:

$$
\begin{aligned}
c &= (-10.5 \quad -7.5 \quad -3.5 \quad -2.5 \quad -1.5 \quad -10) \\
q &= (-0.5 \quad -0.5 \quad -0.5 \quad -0.5 \quad -0.5 \quad 0) \\
a &= (6 \quad 10 \quad 3 \quad 3 \quad 10 \quad 2 \quad 1 \quad 1) \\
lr &= (-1e31 \quad -1e31) \\
ur &= (6.5 \quad 20) \\
lc &= (0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0) \\
uc &= (1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1e31)
\end{aligned}
$$

Note that `a` is the array of nonzero matrix elements (length ne), `ia` is the array of row indices (length ne) and `ja` is the array of column starts (length $n+1$).

The `integer*4` arrays:

$$
\begin{aligned}
ia &= (1 \quad 2 \quad 1 \quad 1 \quad 2 \quad 1 \quad 1 \quad 2) \\
ja &= (1 \quad 3 \quad 4 \quad 6 \quad 7 \quad 8 \quad 9)
\end{aligned}
$$

The subroutine call is:

```
      call barscqp(
     $        lc, uc, n, barspace, work, optfname,
     $        barstatus, zlbest, zub, xbest, totaltime,
     $        proname, c, q, lr, ur, a, ia, ja, m, numint, ne)
```

The screen output for the above input is shown below.

```
================================================================================
                        Welcome to BARON v. 4.0
                 Global Optimization by BRANCH-AND-REDUCE
================================================================================
                  Separable Concave Quadratic Programming
================================================================================
 Starting solution is feasible with a value of      .000000D+00
 Preprocessing found feasible solution with value  -.949306D+02
 Preprocessing found feasible solution with value  -.155962D+03
================================================================================
       We have space for 599988           nodes in the tree
================================================================================

  Itn. no.      Open Nodes      Total Time     Lower Bound      Upper Bound
```

```
*       1              1        000:00:00    -.100000D+52     -.213000D+03
        1              0        000:00:00    -.213000D+03     -.213000D+03


                      *** Successful Termination ***


  Total time elapsed   :     000:00:00,    in seconds :           .01
       on preprocessing:     000:00:00,    in seconds :           .00
       on navigating   :     000:00:00,    in seconds :           .01
       on relaxed      :     000:00:00,    in seconds :           .00
       on local        :     000:00:00,    in seconds :           .00
       on tightening   :     000:00:00,    in seconds :           .00
       on marginals    :     000:00:00,    in seconds :           .00
       on probing      :     000:00:00,    in seconds :           .00


   Total no. of BaR iterations:       1
   Best solution found at node:       1
   Max. no. of nodes in memory:       1

 All done with problem scqp
================================================================================
```

8.3 Module barscp: Separable Concave Programming

The same example as in scqp is solved here.

For type = 0, the problem file is:

```
scp                                                  problem name
2       6       0                                    m, n, numint
0                                                    type
       0    0    0    0    0     0                   lc
-1e31  6    3    3    2    1     0     6.5           lr, a, ur (row 1)
-1e31 10    0   10    0    0     1     20            lr, a, ur (row 2)
       1    1    1    1    1     1e31                uc
```

For type = 1, the problem file is:

```
scp                                                    problem name
2       6       0                                      m, n, numint
1                                                      type
8                                                      ne
6   10    3    3   10    2    1    1                  a(i),  i=1,ne
1    2    1    1    2    1    1    2                  ia(i), i=1,ne
1    3    4    6    7    8    9                       ja(i), i=1,n+1
-1e31   -1e31                                         lr(i), i=1,m
  6.5      20                                         ur(i), i=1,m
       0    0    0    0    0     0                   lc
       1    1    1    1    1     1e31                uc
```

For either value of `type`, a starting point may be optionally supplied by providing an array of length `n` in the last row of the input file.

Problem data can also be passed directly in memory by calling the subroutine `barscp` as shown below. The arrays should be declared as follows:

The `real*8` arrays:

$$
\begin{aligned}
a &= (6 \quad 10 \quad 3 \quad 3 \quad 10 \quad 2 \quad 1 \quad 1) \\
lr &= (-1e31 \quad -1e31) \\
ur &= (6.5 \quad 20) \\
lc &= (0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0) \\
uc &= (1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1e31)
\end{aligned}
$$

The `integer*4` arrays:

$$
\begin{aligned}
ia &= (1 \quad 2 \quad 1 \quad 1 \quad 2 \quad 1 \quad 1 \quad 2) \\
ja &= (1 \quad 3 \quad 4 \quad 6 \quad 7 \quad 8 \quad 9)
\end{aligned}
$$

The subroutine call is:

```
      call barscp(
     $        lc, uc, n, barspace, work, optfname,
     $        barstatus, zlbest, zub, xbest, totaltime,
     $        proname, lr, ur, a, ia, ja, m, numint, ne)
```

The user is expected to write a function to provide the univariate concave functions for the objective function. The layout of the required function is given below:

```
*     this function provides the objective function term
*     corresponding to the i-th variable, for this variable
*     being equal to x.

      real*8 function f(i, x)
      implicit none
      integer*4 i
      real*8 x

c     calculation of ''f'' here

      return
      end
```

For the above example, the univariate concave function was computed as $f_i(x_i) = cx_i + qx_i^2$, where c and q given below are used from the `problem` file of `scqp`:

```
      -10.5 -7.5 -3.5 -2.5 -1.5 -10              c
      -.5   -.5  -.5  -.5  -.5    0              q
```

The screen output for the above input is shown below.

```
===========================================================================
                        Welcome to BARON v. 4.0
                  Global Optimization by BRANCH-AND-REDUCE
===========================================================================
                       Separable Concave Programming
===========================================================================
 Preprocessing found feasible solution with value   .000000D+00
 Preprocessing found feasible solution with value  -.949306D+02
 Preprocessing found feasible solution with value  -.155962D+03
===========================================================================
       We have space for  39988           nodes in the tree
===========================================================================

  Itn. no.      Open Nodes      Total Time     Lower Bound      Upper Bound

*       1              1        000:00:00    -.100000D+52     -.213000D+03
        1              0        000:00:00    -.213000D+03     -.213000D+03


                      *** Successful Termination ***
```

```
 Total time elapsed   :     000:00:00,    in seconds :          .00
      on preprocessing:     000:00:00,    in seconds :          .00
      on navigating   :     000:00:00,    in seconds :          .00
      on relaxed      :     000:00:00,    in seconds :          .00
      on local        :     000:00:00,    in seconds :          .00
      on tightening   :     000:00:00,    in seconds :          .00
      on marginals    :     000:00:00,    in seconds :          .00
      on probing      :     000:00:00,    in seconds :          .00


  Total no. of BaR iterations:       1
  Best solution found at node:       1
  Max. no. of nodes in memory:       1

All done with problem scp
================================================================================
```

8.4 Module `barpes`: Power Economies of Scale

$$
\begin{aligned}
\min \quad & x_1^{0.6} + x_2^{0.6} + x_3^{0.4} + 2x_4 + 5x_5 - x_6 - 4x_7 \\
\text{s.t.} \quad & -3x_1 + x_2 - 3x_4 = 0 \\
& -2x_2 + x_3 - 2x_5 = 0 \\
& 4x_4 - x_6 = 0 \\
& x_1 + 2x_4 \leq 4 \\
& x_2 + x_5 \leq 4 \\
& x_3 + x_6 \leq 6 \\
& x_3 - x_7 = 0 \\
& 0 \leq x_1 \leq 3 \\
& 0 \leq x_2 \leq 4 \\
& 0 \leq x_3 \leq 4 \\
& 0 \leq x_4 \leq 2 \\
& 0 \leq x_5 \leq 2 \\
& 0 \leq x_6 \leq 6 \\
& 0 \leq x_7 \leq 4
\end{aligned}
$$

The input file for the above example problem is as follows:
For `type` = 0, the problem file is:

```
pes                                                    problem name
7       7       0                                      m, n, numint
0                                                      type
         0    0    0    0    0   0   0         0      lc
0       -3    1    0   -3    0   0   0   0     0      lr, a, ur (row 1)
0        0   -2    1    0   -2   0   0   0     0      lr, a, ur (row 2)
0        0    0    0    4    0  -1   0   0     0      lr, a, ur (row 3)
-1e31    1    0    0    2    0   0   0   4     0      lr, a, ur (row 4)
-1e31    0    1    0    0    1   0   0   4     0      lr, a, ur (row 5)
-1e31    0    0    1    0    0   1   0   6     0      lr, a, ur (row 6)
0        0    0    1    0    0   0  -1   0     0      lr, a, ur (row 7)
         3    4    4    2    2   6   4         0      uc
         1    1    1    2    5  -1  -4         0      c
         0.6  0.6  0.4  1    1   1   1         0      q
```

For `type` = 1, the problem file is:

```
pes                                                    problem name
7       7       0                                      m, n, numint
1                                                      type
16                                                     ne
-3  1  1  -2  1  1  1  1  -3  4  2  -2  1  -1  1  -1   a(i),  i=1,ne
 1  4  1   2  5  2  6  7   1  3  4   2  5   3  6   7   ia(i), i=1,ne
 1  3  6   9 12 14 16 17                              ja(i), i=1,n+1
0     0     0 -1e31 -1e31 -1e31     0                 lr(i), i=1,m
0     0     0     4     4     6     0                 ur(i), i=1,m
0     0     0     0     0     0     0                 lc(i), i=1,n
3     4     4     2     2     6     4                 uc(i), i=1,n
1     1     1     2     5    -1    -4                 c(i),  i=1,n
0.6 0.6   0.4     1     1     1     1                 q(i),  i=1,n
```

For either value of `type`, a starting point may be optionally supplied by providing an array of length `n` in the last row of the input file.

Problem data can also be passed directly in memory by calling the subroutine `barpes` as shown below. The arrays should be declared as follows:

The `real*8` arrays:

$$
\begin{aligned}
c &= (1 \quad 1 \quad 1 \quad 2 \quad 5 \quad -1 \quad -4) \\
q &= (0.6 \quad 0.6 \quad 0.4 \quad 1 \quad 1 \quad 1 \quad 1) \\
a &= (-3 \quad 1 \quad 1 \quad -2 \quad 1 \quad 1 \quad 1 \quad 1 \quad -3 \quad 4 \quad 2 \quad -2 \quad 1 \quad -1 \quad 1 \quad -1) \\
lr &= (0 \quad 0 \quad 0 \quad -1e31 \quad -1e31 \quad -1e31 \quad 0)
\end{aligned}
$$

$$
\begin{aligned}
ur &= (0 \quad 0 \quad 0 \quad 4 \quad 4 \quad 6 \quad 0) \\
lc &= (0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0) \\
uc &= (3 \quad 4 \quad 4 \quad 2 \quad 2 \quad 6 \quad 4)
\end{aligned}
$$

The `integer*4` arrays:

$$
\begin{aligned}
ia &= (1 \quad 4 \quad 1 \quad 2 \quad 5 \quad 6 \quad 7 \quad 1 \quad 3 \quad 4 \quad 2 \quad 5 \quad 3 \quad 6 \quad 7) \\
ja &= (1 \quad 3 \quad 6 \quad 9 \quad 12 \quad 14 \quad 16 \quad 17)
\end{aligned}
$$

The subroutine call is:

```
      call barpes(
     $        lc, uc, n, barspace, work, optfname,
     $        barstatus, zlbest, zub, xbest, totaltime,
     $        proname, c, q, lr, ur, a, ia, ja, m, numint, ne)
```

The above input gives the following screen output.

```
================================================================================
                          Welcome to BARON v. 4.0
                    Global Optimization by BRANCH-AND-REDUCE
================================================================================
                       Programming with Economies of Scale
================================================================================
 Starting solution is feasible with a value of       .000000D+00
 Preprocessing found feasible solution with value  -.119591D+02
================================================================================
       We have space for 555543           nodes in the tree
================================================================================

  Itn. no.      Open Nodes      Total Time     Lower Bound      Upper Bound

*       1               1        000:00:00    -.100000D+52     -.134019D+02
        1               1        000:00:00    -.134031D+02     -.134019D+02
        2               2        000:00:00    -.134031D+02     -.134019D+02
        3               1        000:00:00    -.134019D+02     -.134019D+02
        4               1        000:00:00    -.134019D+02     -.134019D+02
        5               0        000:00:00    -.134019D+02     -.134019D+02


                       *** Successful Termination ***


  Total time elapsed   :     000:00:00,    in seconds :           .03
       on preprocessing:     000:00:00,    in seconds :           .02
```

```
      on navigating   :     000:00:00,    in seconds :          .00
      on relaxed      :     000:00:00,    in seconds :          .01
      on local        :     000:00:00,    in seconds :          .00
      on tightening   :     000:00:00,    in seconds :          .00
      on marginals    :     000:00:00,    in seconds :          .00
      on probing      :     000:00:00,    in seconds :          .00


  Total no. of BaR iterations:       5
  Best solution found at node:       1
  Max. no. of nodes in memory:       2

 All done with problem pes
================================================================================
```

8.5 Module `barfcp`: Fixed Charge Programming

$$
\begin{aligned}
\min \quad & f(x) = \sum_{i=1}^{3} f_i(x_i) \\
\text{s.t.} \quad & 3x_1 + 2x_2 + 6x_3 \leq 150 \\
& 4x_1 + 3x_2 + 4x_3 \leq 160 \\
& x_i \geq 0, i = 1, \ldots, 3
\end{aligned}
$$

$$
f_1(x_1) = \begin{cases} 200 - 6x_1, & \text{when } x_1 > 0 \\ 0, & \text{when } x_1 = 0 \end{cases}
$$

$$
f_2(x_2) = \begin{cases} 150 - 4x_2, & \text{when } x_2 > 0 \\ 0, & \text{when } x_2 = 0 \end{cases}
$$

$$
f_3(x_3) = \begin{cases} 100 - 7x_3, & \text{when } x_3 > 0 \\ 0, & \text{when } x_3 = 0 \end{cases}
$$

The above example can be input in the following form:

For `type` $= 0$, the problem file is:

```
fcp                                  problem name
2       3       0                    m, n, numint
0                                    type
         0    0    0                 lc(i), i=1,n
-1e31    3    2    6    150          lr, a, ur (row 1)
-1e31    4    3    4    160          lr, a, ur (row 2)
        1e31  1e31 1e31               uc(i), i=1,n
        200  150   100                c(i),  i=1,n
        -6   -4   -7                  q(i),  i=1,n
```

For `type` = 1, the problem file is:

```
fcp                                  problem name
2       3       0                    m, n, numint
1                                    type
6                                    ne
3   4   2   3   6   4                a(i),  i=1,ne
1   2   1   2   1   2                ia(i), i=1,ne
1   3   5   7                        ja(i), i=1,n+1
-1e31   -1e31                        lr(i), i=1,m
  150     160                        ur(i), i=1,m
    0       0       0                lc(i), i=1,n
 1e31    1e31    1e31                uc(i), i=1,n
  200     150     100                c(i),  i=1,n
   -6      -4      -7                q(i),  i=1,n
```

For either value of `type`, a starting point may be optionally supplied by providing an array of length `n` in the last row of the input file.

Problem data can also be passed directly in memory by calling the subroutine `barfcp` as shown below. The arrays should be declared as follows:

The `real*8` arrays:

$$
\begin{aligned}
c &= (\,200 \quad 150 \quad 100\,) \\
q &= (\,-6 \quad -4 \quad -7\,) \\
a &= (\,3 \quad 4 \quad 2 \quad 3 \quad 6 \quad 4\,) \\
lr &= (\,-1e31 \quad -1e31\,) \\
ur &= (\,150 \quad 160\,) \\
lc &= (\,0 \quad 0 \quad 0\,) \\
uc &= (\,1e31 \quad 1e31 \quad 1e31\,)
\end{aligned}
$$

The `integer*4` arrays:

$$\begin{aligned} ia &= (1 \quad 2 \quad 1 \quad 2 \quad 1 \quad 2) \\ ja &= (1 \quad 3 \quad 5 \quad 7) \end{aligned}$$

The subroutine call is:

```
      call barfcp(
     $        lc, uc, n, barspace, work, optfname,
     $        barstatus, zlbest, zub, xbest, totaltime,
     $        proname, c, q, lr, ur, a, ia, ja, m, numint, ne)
```

The above input gives the screen output given below:

```
================================================================================
                        Welcome to BARON v. 4.0
                 Global Optimization by BRANCH-AND-REDUCE
================================================================================
                        Fixed-Charge Programming
================================================================================
 Starting solution is feasible with a value of      .000000D+00
 Preprocessing found feasible solution with value  -.400000D+02
 Preprocessing found feasible solution with value  -.633333D+02
 Preprocessing found feasible solution with value  -.750000D+02
================================================================================
       We have space for 789461           nodes in the tree
================================================================================

  Itn. no.      Open Nodes      Total Time     Lower Bound      Upper Bound

        1              1        000:00:00    -.100000D+52     -.750000D+02
        1              1        000:00:00    -.817500D+02     -.750000D+02
        2              1        000:00:00    -.817500D+02     -.750000D+02
        3              0        000:00:00    -.750000D+02     -.750000D+02


                     *** Successful Termination ***


  Total time elapsed   :     000:00:00,    in seconds :           .03
       on preprocessing:     000:00:00,    in seconds :           .02
       on navigating   :     000:00:00,    in seconds :           .00
       on relaxed      :     000:00:00,    in seconds :           .01
       on local        :     000:00:00,    in seconds :           .00
       on tightening   :     000:00:00,    in seconds :           .00
       on marginals    :     000:00:00,    in seconds :           .00
```

```
      on probing      :     000:00:00,    in seconds :          .00


   Total no. of BaR iterations:       3
   Best solution found at node:       0
   Max. no. of nodes in memory:       2

 All done with problem fcp
================================================================================
```

8.6 Module `barfp`: Fractional Programming

$$
\begin{aligned}
\min \quad & \frac{1.5x_1 + 2x_2 + 2.5x_3 + 3.5x_4 + 3x_5 + 2.5x_6 + 1.5x_7}{x_1 + 2x_2 + x_3 + 4x_4 + 2.5x_5 + 2x_6 + 3x_7} \\
\text{s.t.} \quad & 2x_1 + x_2 + 2x_3 + 3x_4 + 2.5x_5 + 1.5x_6 + 3x_7 \leq 30 \\
& 2x_1 + 3x_3 + 1.5x_5 + 2x_6 + x_7 \leq 15 \\
& -100 \leq -3x_1 + 6x_2 + 3x_3 + x_4 + 4x_5 + x_6 \leq 20 \\
& 3 \leq x_2 + 2x_4 + 6x_5 + 2x_6 + x_7 \leq 18 \\
& -10 \leq x_1 - 2x_2 + 2x_3 - 4x_4 - 3x_5 + 5x_6 + 3x_7 \leq 20 \\
& -3.5 \leq 2.5x_1 - 3.5x_2 + 1.5x_3 + 2.25x_4 \\
& \qquad\qquad -3.5x_5 + 1.25x_6 + 2x_7 \leq 20 \\
& -10 \leq -1.5x_1 + 0.5x_2 - 1.5x_3 + x_4 - 3x_5 + 2.5x_6 - x_7 \leq 5 \\
& 1.5 \leq x_1 \leq 7 \\
& -1.5 \leq x_2 \leq 7 \\
& 1 \leq x_3 \leq 7 \\
& 0 \leq x_4 \leq 7 \\
& 0 \leq x_5 \leq 7 \\
& 0 \leq x_6 \leq 7 \\
& 0 \leq x_7 \leq 7 \\
& x_i \text{ integer for } i = 1, \ldots, 7
\end{aligned}
$$

The above example can be input in any of the following two forms.

For `type` = `0`, the problem file is:

```
fp                                                    problem name
7       7       7                                     m, n, numint
0                                                     type
        1.5 -1.5  1   0     0   0    0                lc(i), i=1,n
-1.0E31 2    1    2   3     2.5 1.5  3     30         lr, a, ur (row 1)
-1.0E31 2    0    3   0     1.5 2    1     15         lr, a, ur (row 2)
-100    -3   6    3   1     4   1    0     20         lr, a, ur (row 3)
3       0    1    0   2     6   2    1     18         lr, a, ur (row 4)
-10     1   -2    2  -4    -3   5    3     20         lr, a, ur (row 5)
-3.5    2.5 -3.5  1.5 2.25 -3.5 1.25 2     20         lr, a, ur (row 6)
-10    -1.5 0.5  -1.5 1.   -3   2.5 -1.    5          lr, a, ur (row 7)
        7    7    7   7     7   7    7                uc(i), i=1,n
        1.5  2    2.5 3.5   3   2.5  1.5              c(i),  i=1,n
        1    2    1   4     2.5 2    3                q(i),  i=1,n
0   0                                                 alpha, beta
```

For `type` = 1, the problem file is:

```
fp                                                    problem name
7       7       7                                     m, n, numint
1                                                     type
44                                                    ne
2.0 2.0 -3.0 1.0 2.5 -1.5 1.0 6.0 1.0 -2.0 -3.5                  \
0.5 2.0 3.0 3.0 2.0 1.5 -1.5 3.0 1.0 2.0 -4.0                    \
2.25 1.0 2.5 1.5 4.0 6.0 -3.0 -3.5 -3.0 1.5 2.0                  \
1.0 2.0 5.0 1.25 2.5 3.0 1.0 1.0 3.0 2.0 -1.0                    \
1 2 3 5 6 7 1 3 4 5 6 7 1 2 3 5 6 7 1 3 4 5 6 7     a(i),  i=1,ne
1 2 3 4 5 6 7 1 2 3 4 5 6 7 1 2 4 5 6 7             ia(i), i=1,ne
1 7 13 19 25 32 39 45                               ja(i), i=1,n+1
    -1.0E31 -1.0E31   -100 3 -10 -3.5 -10           lr(i), i=1,m
    30   15   20  18    20  20   5                  ur(i), i=1,m
        1.5 -1.5  1   0     0   0    0              lc(i), i=1,n
        7    7    7   7     7   7    7              uc(i), i=1,n
        1.5  2    2.5 3.5   3   2.5  1.5            c(i),  i=1,n
        1    2    1   4     2.5 2    3              q(i),  i=1,n
0   0                                               alpha, beta
```

For either value of `type`, a starting point may be optionally supplied by providing an array of length `n` in the last row of the input file.

Problem data can also be passed directly in memory by calling the subroutine `barfp` as shown below. The arrays should be declared in the same fashion as before. Here `alpha` = 0.0 and `beta` = 0.0 are `real*8` values.

The subroutine call is:

```
      call barfp(
     $     lc, uc, n, barspace, work, optfname,
     $     barstatus, zlbest, zub, xbest, totaltime,
     $     proname, c, q, alpha, beta, lr, ur,
     $     a, ia, ja, m, numint, ne)
```

The above input gives the screen output given below:

```
===============================================================================
                        Welcome to BARON v. 4.0
                  Global Optimization by BRANCH-AND-REDUCE
===============================================================================
                          Fractional Programming
===============================================================================
===============================================================================
  Itn. no.      Open Nodes      Total Time     Lower Bound      Upper Bound

        1               1        000:00:00    -.100000D+52      .100000D+52
*       1               1        000:00:00     .706522D+00      .720000D+00
*       1               0        000:00:00     .690476D+00      .690476D+00
        1               0        000:00:00     .690476D+00      .690476D+00


                     *** Successful Termination ***


  Total time elapsed   :     000:00:00,    in seconds :           .12
       on preprocessing:     000:00:00,    in seconds :           .10
       on navigating   :     000:00:00,    in seconds :           .00
       on relaxed      :     000:00:00,    in seconds :           .02
       on local        :     000:00:00,    in seconds :           .00
       on tightening   :     000:00:00,    in seconds :           .00
       on marginals    :     000:00:00,    in seconds :           .00
       on probing      :     000:00:00,    in seconds :           .00


   Total no. of BaR iterations:       1
   Best solution found at node:       1
   Max. no. of nodes in memory:       1

 All done with problem fp
===============================================================================
```

8.7 Module `barpoly`: Univariate Unconstrained Polynomials

$$\begin{aligned} \min \quad & 0.1 - x - 3.95x^2 + 7.1x^3 + 0.4875x^4 - 2.08x^5 + \frac{1}{6}x^6 \\ \text{s.t.} \quad & -2 \leq x \leq 11 \\ & x \in \mathbb{R} \end{aligned}$$

The above example can be input in the following form:

```
poly            problem name
6               noterms (degree of polynomial)
0               numint (1 if integer solution required, 0 else)
-2.             lc
11.             uc
0.1             objadd
-1.             c1
-3.95           c2
7.1             c3
0.4875          c4
-2.08           c5
.16666666666666 c6
```

Note that `type` is not required here.

A starting point may be optionally supplied in the last row of the input file.

Problem data can also be passed directly in memory by calling the subroutine `barpoly` as shown below. The arrays should be declared in the same fashion as before. Here, `objadd` $= 0.1$ is the constant term in the objective. The degree of the polynomial in the objective is `noterms` $= 6$. The `real*8` array `coeff` represents the coefficients of the individual terms in the objective. In this way, `coeff`(i) is the coefficient of x^i in the objective. If x^i does not appear in the objective and $1 \leq i \leq$ `noterms`, `coeff`$(i) = 0.0$. Note that $n = 1$ in `barpoly`. If integer solution is required, `numint` is set to 1.

The subroutine call is:

```
      call barpoly(
     $     lc, uc, n, barspace, work, optfname,
     $     barstatus, zlbest, zub, xbest, totaltime,
     $     proname, coeff, objadd, numint, noterms)
```

The above input gives the screen output given below:

```
================================================================================
                          Welcome to BARON v. 4.0
                    Global Optimization by BRANCH-AND-REDUCE
================================================================================
                Univariate Unconstrained Polynomial Programming
================================================================================
 Starting solution is feasible with a value of       .100000D+00
================================================================================
  Itn. no.      Open Nodes      Total Time     Lower Bound      Upper Bound

         1               1        000:00:00    -.100000D+52      .100000D+00
         1               1        000:00:00    -.100000D+52      .100000D+00
*        2               2        000:00:00    -.100000D+52     -.107819D+02
*        2               2        000:00:00    -.100000D+52     -.108948D+02
*        2               2        000:00:00    -.100000D+52     -.111060D+02
*        2               2        000:00:00    -.100000D+52     -.112663D+02
*        2               2        000:00:00    -.100000D+52     -.113515D+02
         2               2        000:00:00    -.100000D+52     -.113515D+02
*        3               1        000:00:00    -.100000D+52     -.173611D+05
         3               1        000:00:00    -.195184D+06     -.173611D+05
*        4               2        000:00:00    -.195184D+06     -.249521D+05
*        4               2        000:00:00    -.195184D+06     -.270831D+05
*        4               2        000:00:00    -.195184D+06     -.279032D+05
         4               1        000:00:00    -.195184D+06     -.279032D+05
*        5               1        000:00:00    -.195184D+06     -.296472D+05
         5               1        000:00:00    -.452908D+05     -.296472D+05
*        6               2        000:00:00    -.452908D+05     -.297109D+05
         6               2        000:00:00    -.452908D+05     -.297109D+05
         7               1        000:00:00    -.313699D+05     -.297109D+05
         8               1        000:00:00    -.313699D+05     -.297109D+05
*        9               1        000:00:00    -.313699D+05     -.297617D+05
         9               1        000:00:00    -.301048D+05     -.297617D+05
*       10               2        000:00:00    -.301048D+05     -.297619D+05
*       10               2        000:00:00    -.301048D+05     -.297624D+05
        10               2        000:00:00    -.301048D+05     -.297624D+05
        11               2        000:00:00    -.297867D+05     -.297624D+05
        12               3        000:00:00    -.297867D+05     -.297624D+05
*       13               2        000:00:00    -.297867D+05     -.297632D+05
        13               2        000:00:00    -.297682D+05     -.297632D+05
        14               3        000:00:00    -.297682D+05     -.297632D+05
        15               3        000:00:00    -.297660D+05     -.297632D+05
        16               3        000:00:00    -.297660D+05     -.297632D+05
        17               3        000:00:00    -.297660D+05     -.297632D+05
*       18               3        000:00:00    -.297660D+05     -.297632D+05
        18               3        000:00:00    -.297660D+05     -.297632D+05
        19               2        000:00:00    -.297633D+05     -.297632D+05
        20               3        000:00:00    -.297633D+05     -.297632D+05
        21               3        000:00:00    -.297633D+05     -.297632D+05
        22               3        000:00:00    -.297633D+05     -.297632D+05
```

```
       23              2        000:00:00    -.297633D+05     -.297632D+05
       24              2        000:00:00    -.297633D+05     -.297632D+05
       25              2        000:00:00    -.297632D+05     -.297632D+05
       26              2        000:00:00    -.297632D+05     -.297632D+05
*      27              2        000:00:00    -.297632D+05     -.297632D+05
       27              2        000:00:00    -.297632D+05     -.297632D+05
       28              1        000:00:00    -.297632D+05     -.297632D+05
       29              2        000:00:00    -.297632D+05     -.297632D+05
       30              2        000:00:00    -.297632D+05     -.297632D+05
       31              2        000:00:00    -.297632D+05     -.297632D+05
*      32              2        000:00:00    -.297632D+05     -.297632D+05
       32              2        000:00:00    -.297632D+05     -.297632D+05
       33              3        000:00:00    -.297632D+05     -.297632D+05
       34              4        000:00:00    -.297632D+05     -.297632D+05
       35              3        000:00:00    -.297632D+05     -.297632D+05
       36              3        000:00:00    -.297632D+05     -.297632D+05
*      37              3        000:00:00    -.297632D+05     -.297632D+05
       37              3        000:00:00    -.297632D+05     -.297632D+05
       38              2        000:00:00    -.297632D+05     -.297632D+05
*      39              3        000:00:00    -.297632D+05     -.297632D+05
       39              3        000:00:00    -.297632D+05     -.297632D+05
       40              3        000:00:00    -.297632D+05     -.297632D+05
       41              2        000:00:00    -.297632D+05     -.297632D+05
*      42              2        000:00:00    -.297632D+05     -.297632D+05
       42              1        000:00:00    -.297632D+05     -.297632D+05
       43              1        000:00:00    -.297632D+05     -.297632D+05
       44              0        000:00:00    -.297632D+05     -.297632D+05


                    *** Successful Termination ***


  Total time elapsed   :     000:00:00,    in seconds :          .04
       on preprocessing:     000:00:00,    in seconds :          .01
       on navigating   :     000:00:00,    in seconds :          .02
       on relaxed      :     000:00:00,    in seconds :          .00
       on local        :     000:00:00,    in seconds :          .00
       on tightening   :     000:00:00,    in seconds :          .01
       on marginals    :     000:00:00,    in seconds :          .00
       on probing      :     000:00:00,    in seconds :          .00


    Total no. of BaR iterations:      44
    Best solution found at node:      42
    Max. no. of nodes in memory:       4

 All done with problem poly
==============================================================================
```

8.8 Module `barlmp`: Linear Multiplicative Programming

$$
\begin{aligned}
\min \quad & (2 + 7x_1 + 9x_2 + 8x_3 + 8x_4) \times (9 + 9x_2 + 5x_3 + 4x_4) \\
& \times (6 + 3x_1 + 8x_2 + 3x_3 + 8x_4) \\
\text{s.t.} \quad & -88x_1 - x_2 - 15x_3 - 82x_4 \leq -6 \\
& -30x_1 - 25x_2 - 83x_3 - 68x_4 \leq -6 \\
& -59x_1 - 72x_3 - 29x_4 \leq -3 \\
& -17x_1 - 35x_2 - 57x_3 - 24x_4 \leq -3 \\
& x_i \text{ integer for } i = 1, \ldots, 3 \\
& 0 \leq x_i \leq 100, i = 1, \ldots, 3
\end{aligned}
$$

The above example can be input in any of the following two forms.

For `type` = `0`, the problem file is:

```
lmp                                          problem name
4 4 3 3                                      m, n, numint, np
0                                            type
          0     0     0     0                lc(i), i=1,n
-1.e31  -88    -1   -15   -82   -6          lr,a,ur (row 1)
-1.e31  -30   -25   -83   -68   -6          lr,a,ur (row 2)
-1.e31  -59     0   -72   -29   -3          lr,a,ur (row 3)
-1.e31  -17   -35   -57   -24   -3          lr,a,ur (row 4)
        100   100   100   100               uc(i), i=1,n
2       7     9     8     8                 const, cij (product term 1)
9       0     9     5     4                 const, cij (product term 1)
6       3     8     3     8                 const, cij (product term 1)
```

For `type` = `1`, the problem file is:

```
lmp                                                          problem name
4 4 3 3                                                      m,n,numint,np
1                                                                 type
15                                                                ne
-88 -30 -59 -17 -1 -25 -35 -15 -83 -72 -57 -82 -68 -29 -24   a
1 2 3 4 1 2 4 1 2 3 4 1 2 3 4                                     ia
1 5 8 12 16                                                       ja
-1.e31   -1.e31   -1.e31   -1.e31                                 lr
-6       -6       -3       -3                                    ur
 0        0        0         0                                   lc
 1.e31    1.e31    1.e31     1.e31                               uc
2   7     9     8     8                                    const, cij
9         9     5     4
6   3     8     3     8
```

For either value of `type`, a starting point may be optionally supplied by providing an array of length `n` in the last row of the input file.

Problem data can also be passed directly in memory by calling the subroutine `barlmp` as shown below. The arrays should be declared in the same fashion as before. The first `numint` variables are considered integers. The number of linear terms in the objective product is denoted by `np` which is declared as `integer*4`. The `real*8` array `const` is of length `maxnp` and the `real*8` two dimensional array `cij` is of length `(maxnp, maxn)`. `const`(i) contains the constant term of the ith product of the objective function. `cij`(i, j) contains the coefficient of the jth variable in the ith linear term of the objective. In this example,

$$\texttt{const} = \begin{pmatrix} 2 & 9 & 6 \end{pmatrix}$$

$$\texttt{cij} = \begin{pmatrix} 7 & 9 & 8 & 8 \\ 0 & 9 & 5 & 4 \\ 3 & 8 & 3 & 8 \end{pmatrix}$$

The subroutine call is:

```
      call barlmp(
     $        lc, uc, n, barspace, work, optfname,
     $        barstatus, zlbest, zub, xbest, totaltime,
     $        proname, const, cij, lr, ur, a, ia, ja, m, np,
     $        numint, ne, maxnp)
```

The above input gives the screen output given below:

```
================================================================================
                          Welcome to BARON v. 4.0
                    Global Optimization by BRANCH-AND-REDUCE
================================================================================
                       Linear Multiplicative Programming
================================================================================
 Preprocessing found feasible solution with value   .127781D+11
 Preprocessing found feasible solution with value   .599288D+10
 Preprocessing found feasible solution with value   .193331D+07
================================================================================
  Itn. no.      Open Nodes      Total Time     Lower Bound      Upper Bound

        1               1        000:00:00    -.100000D+52      .193331D+07
*       1               1        000:00:00     .155679D+03      .199500D+03
        1               0        000:00:00     .199500D+03      .199500D+03


                     *** Successful Termination ***


  Total time elapsed   :     000:00:00,    in seconds :           .02
       on preprocessing:     000:00:00,    in seconds :           .02
       on navigating   :     000:00:00,    in seconds :           .00
       on relaxed      :     000:00:00,    in seconds :           .00
       on local        :     000:00:00,    in seconds :           .00
       on tightening   :     000:00:00,    in seconds :           .00
       on marginals    :     000:00:00,    in seconds :           .00
       on probing      :     000:00:00,    in seconds :           .00


   Total no. of BaR iterations:       1
   Best solution found at node:       1
   Max. no. of nodes in memory:       1

 All done with problem lmp
================================================================================
```

8.9 Module barglmp: General Linear Multiplicative Programming

$$
\begin{aligned}
\min \quad & x_1 + (5 + x_1 - x_2) \times (-1 + x_1 + x_2) \\
\text{s.t.} \quad & 2x_1 + 3x_2 \geq 9 \\
& 3x_1 - x_2 \leq 8 \\
& -x_1 + 2x_2 \leq 8
\end{aligned}
$$

$$x_1 + 2x_2 \leq 12$$
$$0 \leq x_1 \leq 10$$
$$0 \leq x_2 \leq 10$$

The above example can be input in any of the following two forms.

For `type` = `0`, the problem file is:

```
glmp                                    problem name
4       2       0       2               m, n, numint, nt
0                                       type
1       2                               np(i), i=1,nt
        0       0                       lc(i), i=1,n
9       2       3       1.e+15          lr,a,ur (row 1)
-1.e+15 3       -1      8               lr,a,ur (row 2)
-1.e+15 -1      2       8               lr,a,ur (row 3)
-1.e+15 1       2       12              lr,a,ur (row 4)
        10      10                      uc(i), i=1,n
0       1       0                       c0, cij (term wise,
5       1       -1                      and within term,
-1      1       1                       product wise)
```

For `type` = `1`, the problem file is:

```
glmp                                    problem name
4       2       0       2               m, n, numint, nt
1                                       type
1       2                               np(i), i=1,nt
8                                       ne
2 3 -1 1 3 -1 2 2                       a(i),  i=1,ne
1 2 3 4 1 2 3 4                         ia(i), i=1,ne
1 5 9                                   ja(i), i=1,n+1
        9 -1.e+15 -1.e+15  -1.e+15      lr(i), i=1,m
   1.e+15       8       8       12      ur(i), i=1,m
        0       0                       lc(i), i=1,n
        10      10                      uc(i), i=1,n
0       1       0                       c0, cij (term wise,
5       1       -1                      and within term,
-1      1       1                       product wise)
```

For either value of `type`, a starting point may be optionally supplied by providing an array of length `n` in the last row of the input file.

Problem data can also be passed directly in memory by calling the subroutine `barglmp` as shown below. The arrays should be declared in the same fashion as before. The first `numint` variables are considered integers. The number of product terms in the objective is given by the `integer*4`, `nt`. The `integer*4` array `np` is of length `nt`. `np`(i) is the number of linear terms in the ith product term of the objective.

The `real*8` array `c0` is of length `maxmp` and the `real*8` two dimensional array `cij` is of length `(maxmp, maxn)`. The `integer*4`, `mp` is the total number of linear terms in the objective. `c0`(i) contains the constant term of the ith linear term of the objective function. `cij`(i, j) contains the coefficient of the jth variable in the ith linear term of the objective function. In this example,

$$\begin{aligned} c0 &= \begin{pmatrix} 0 & 5 & -1 \end{pmatrix} \\ cij &= \begin{pmatrix} 1 & 0 \\ 1 & -1 \\ 1 & 1 \end{pmatrix} \end{aligned}$$

The subroutine call is:

```
      call barglmp(
     $     lc, uc, n, barspace, work, optfname,
     $     barstatus, zlbest, zub, xbest, totaltime,
     $     proname, c0, cij, lr, ur, a, ia, ja, m, np, mp,
     $     row, col, numint, ne, nt, maxn, maxmp)
```

The above input gives the screen output given below:

```
================================================================================
                        Welcome to BARON v. 4.0
                  Global Optimization by BRANCH-AND-REDUCE
================================================================================
                 General Linear Multiplicative Programming
================================================================================
 Preprocessing found feasible solution with value   .150000D+01
 Preprocessing found feasible solution with value  -.250000D+01
================================================================================
  Itn. no.      Open Nodes      Total Time     Lower Bound      Upper Bound
        1               1        000:00:00    -.100000D+52     -.250000D+01
        1               1        000:00:00    -.300000D+01     -.250000D+01
        2               1        000:00:00    -.300000D+01     -.250000D+01
        3               1        000:00:00    -.250171D+01     -.250000D+01
        4               1        000:00:00    -.250171D+01     -.250000D+01
        5               0        000:00:00    -.250000D+01     -.250000D+01
```

```
                    *** Successful Termination ***


 Total time elapsed   :     000:00:00,    in seconds :          .03
      on preprocessing:     000:00:00,    in seconds :          .03
      on navigating   :     000:00:00,    in seconds :          .00
      on relaxed      :     000:00:00,    in seconds :          .00
      on local        :     000:00:00,    in seconds :          .00
      on tightening   :     000:00:00,    in seconds :          .00
      on marginals    :     000:00:00,    in seconds :          .00
      on probing      :     000:00:00,    in seconds :          .00


  Total no. of BaR iterations:       5
  Best solution found at node:      -1
  Max. no. of nodes in memory:       2

All done with problem glmp
================================================================================
```

8.10 Module `bariqp`: Indefinite Quadratic Programming

$$
\begin{aligned}
\min \quad & 2x_1 - 4x_2 + 8x_3 + 4x_4 + 9x_5 \\
& +3x_6 - x_7 - 2x_8 - 4x_9 + 5x_{10} \\
& +x_1x_6 + x_2x_7 + x_3x_8 + x_4x_9 + x_5x_{10} \\
\text{s.t.} \quad & -8x_1 - 6x_3 + 7x_4 - 7x_5 \leq 1 \\
& -6x_1 + 2x_2 - 3x_3 + 9x_4 - 3x_5 \leq 3 \\
& 6x_1 - 7x_3 - 8x_4 + 2x_5 \leq 5 \\
& -x_1 + x_2 - 8x_3 - 7x_4 - 5x_5 \leq 4 \\
& 4x_1 - 7x_2 + 4x_3 + 5x_4 + x_5 \leq 0 \\
& 5x_7 - 4x_8 + 9x_9 - 7x_{10} \leq 0 \\
& 7x_6 + 4x_7 + 3x_8 + 7x_9 + 5x_{10} \leq 7 \\
& 6x_6 + x_7 - 3x_8 + 8x_9 \leq 3 \\
& -3x_6 + 2x_7 + 7x_8 + x_{10} \leq 6 \\
& -2x_6 - 3x_7 + 8x_8 + 5x_9 - 2x_{10} \leq 2 \\
& 0 \leq x_i \leq 20, \ i = 1, \ldots, 10.
\end{aligned}
$$

The above problem can be entered using `type` = 0, in the following format.

```
iqp                                                    problem name
10 10 0 5                                              m,n,numint,qterms
0                                                      type
         0  0  0  0  0  0  0  0  0  0                  lc(i), i=1,n
-1.d31  -8  0 -6  7 -7  0  0  0  0  0  1               lr,a,ur (row 1)
-1.d31  -6  2 -3  9 -3  0  0  0  0  0  3               lr,a,ur (row 2)
-1.d31   6  0 -7 -8  2  0  0  0  0  0  5               lr,a,ur (row 3)
-1.d31  -1  1 -8 -7 -5  0  0  0  0  0  4               lr,a,ur (row 4)
-1.d31   4 -7  4  5  1  0  0  0  0  0  0               lr,a,ur (row 5)
-1.d31   0  0  0  0  0  0  5 -4  9 -7  0               lr,a,ur (row 6)
-1.d31   0  0  0  0  0  7  4  3  7  5  7               lr,a,ur (row 7)
-1.d31   0  0  0  0  0  6  1 -8  8  0  3               lr,a,ur (row 8)
-1.d31   0  0  0  0  0 -3  2  7  0  1  6               lr,a,ur (row 9)
-1.d31   0  0  0  0  0 -2 -3  8  5 -2  2               lr,a,ur (row 10)
        20 20 20 20 20 20 20 20 20 20                  uc(i), i=1,n
     2 -4  8  4  9  3 -1 -2 -4  5                      c(i),  i=1,n
1  1  1  1  1                                         q(i),  i=1,n
1 2 3 4 5                                              iq(i), i=1,n
6 7 8 9 10                                             jq(i), i=1,n
```

In case the constraint matrix is in the column major form (*i.e.*, `type` = 1), the `problem` file looks as shown below:

```
iqp                                                  problem name
10 10 0 5                                            m,n,numint,qterms
1                                                    type
45                                                   ne
-8 -6 6 -1 4 2 1 -7 -6 -3 -7 -8 4 7 9 -8 -7 5 -7                  \
-3 2 -5 1 7 6 -3 -2 5 4 1 2 -3 -4 3 -8 7 8 9 7 8                  \
5 -7 5 1 -2                                          a(i),  i=1,ne
1 2 3 4 5 2 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 7                   \
8 9 10 6 7 8 9 10 6 7 8 9 10 6 7 8 10 6 7 9 10       ia(i), i=1,ne
1 6 9 14 19 24 28 33 38 42 46                        ja(i), i=1,n+1
 -1.d31 -1.d31 -1.d31 -1.d31 -1.d31 -1.d31                        \
 -1.d31 -1.d31 -1.d31 -1.d31                         lr(i), i=1,m
 1 3 5 4 0 0 7 3 6 2                                 ur(i), i=1,m
 0  0  0  0  0  0  0  0  0  0                        lc(i), i=1,n
20 20 20 20 20 20 20 20 20 20                        uc(i), i=1,n
2 -4  8  4  9  3 -1 -2 -4  5                         c(i),  i=1,n
1  1  1  1  1                                       q(i),  i=1,qterms
1 2 3 4 5                                            iq(i), i=1,qterms
6 7 8 9 10                                           jq(i), i=1,qterms
```

For either value of `type`, a starting point may be optionally supplied by providing an array of length `n` in the last row of the input file.

Problem data can also be passed directly in memory by calling subroutine `bariqp` as shown below. The arrays should first be declared as follows:

The `real*8` arrays (all are one dimensional):

$$
\begin{aligned}
c &= (\,2 \quad -4 \quad 8 \quad 4 \quad 9 \quad 3 \quad -1 \quad -2 \quad -4 \quad 5\,) \\
q &= (\,1 \quad 1 \quad 1 \quad 1 \quad 1\,) \\
a &= \begin{pmatrix} -8 & -6 & 6 & -1 & 4 & 2 & 1 & -7 & -6 & -3 \\ -7 & -8 & 4 & 7 & 9 & -8 & -7 & 5 & -7 & -3 \\ 2 & -5 & 1 & 7 & 6 & -3 & -2 & 5 & 4 & 1 \\ 2 & -3 & -4 & 3 & -8 & 7 & 8 & 9 & 7 & 8 \\ 5 & -7 & 5 & 1 & -2 & & & & & \end{pmatrix} \\
lr &\quad (\,-1.d31 \quad -1.d31\,) \\
ur &\quad (\,60 \quad 60\,) \\
lc &\quad (\,0 \quad 0 \quad 0 \quad 0 \quad 0\,) \\
uc &\quad (\,1 \quad 1 \quad 1.d31 \quad 1.d31 \quad 1.d31\,)
\end{aligned}
$$

The `integer*4` arrays (all are one dimensional):

$$iq \quad (\,1 \quad 2 \quad 3 \quad 4 \quad 5\,)$$

$$
\begin{array}{ll}
jq & (\,6 \quad 7 \quad 8 \quad 9 \quad 10\,) \\
ia & \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 2 & 4 & 5 & 1 & 2 \\ 3 & 4 & 5 & 1 & 2 & 3 & 4 & 5 & 1 & 2 \\ 3 & 4 & 5 & 7 & 8 & 9 & 10 & 6 & 7 & 8 \\ 9 & 10 & 6 & 7 & 8 & 9 & 10 & 6 & 7 & 8 \\ 10 & 6 & 7 & 9 & 10 & & & & & \end{pmatrix} \\
ja & (\,1 \quad 6 \quad 9 \quad 14 \quad 19 \quad 24 \quad 28 \quad 33 \quad 38 \quad 42 \quad 46\,)
\end{array}
$$

Note that `a` is the array of nonzero matrix elements (length `ne`), `ia` is the array of row indices (length ne) and `ja` is the array of column starts (length $n+1$). On the other hand, `q` is the array of quadratic terms in the objective and is input by indices, *i.e.*, `q, iq, jq` are of length `qterms` and the kth element of `q` corresponds to the coefficient of $x_i x_j$ in the objective where i and j are the kth elements of `iq` and `jq`, respectively.

The subroutine call is:

```
      call bariqp(
     $        lc, uc, n, barspace, work, optfname,
     $        barstatus, zlbest, zub, xbest, totaltime,
     $        proname, c, q, iq, jq, lr, ur, a, ia, ja,
     $        m, numint, qterms, ne)
```

The screen output for the above input is shown below. The options used in the run included `diagonalize`= 0 (no rotation of the objective) and `brstra`= 1 (bisection branching strategy).

```
=============================================================================
 Starting solution is feasible with a value of      -.453797D+02
=============================================================================
       We have space for      348820      nodes in the tree
=============================================================================
  Itn. no.      Open Nodes      Total Time     Lower Bound      Upper Bound
*       1               1        000:00:00    -.100000D+52     -.453797D+02
        1               1        000:00:00    -.453799D+02     -.453797D+02
        2               1        000:00:00    -.453799D+02     -.453797D+02
        3               0        000:00:00    -.453797D+02     -.453797D+02


                     *** Successful Termination ***


  Total time elapsed   :     000:00:00,    in seconds :           .05
       on preprocessing:     000:00:00,    in seconds :           .03
```

```
      on navigating   :     000:00:00,    in seconds :          .01
      on relaxed      :     000:00:00,    in seconds :          .01
      on local        :     000:00:00,    in seconds :          .00
      on tightening   :     000:00:00,    in seconds :          .00
      on marginals    :     000:00:00,    in seconds :          .00
      on probing      :     000:00:00,    in seconds :          .00


  Total no. of BaR iterations:       3
  Best solution found at node:       1
  Max. no. of nodes in memory:       1

 All done with problem iqp
================================================================================
```

8.11 Module `barmilp`: Mixed Integer Linear Programming

$$
\begin{aligned}
\min \quad & x_1 + 4x_2 + 9x_3 \\
\text{s.t.} \quad & x_1 + x_2 \leq 5 \\
& -x_1 - x_3 \leq -10 \\
& -x_2 + x_3 = 7 \\
& 0 \leq x_1 \leq 4 \\
& -1 \leq x_2 \leq 1 \\
& x_1, x_2 \text{ integers} \\
& x_3 \geq 0
\end{aligned}
$$

The above example can be input in any of the following two forms.

For `type` = `0`, the problem file is:

```
milp                                 problem name
 3 3 2                               m, n, numint
 0                                   type
 0.0     -1      0.0                 lc(i), i=1,n
 -1.0d31   1   1   0    5            lr,a,ur (row 1)
      10   1   0   1    1.0d31       lr,a,ur (row 2)
       7   0  -1   1    7            lr,a,ur (row 3)
 4       1       1d31                uc(i), i=1,n
 1       4       9                   c(i),  i=1,n
```

For `type` = `1`, the problem file is:

```
milp                                  problem name
 3 3 2                                m, n, numint
 1                                    type
 6                                    ne
 1 1 1 -1 1 1                         a(i),  i=1,ne
 1 2 1 3 2 3                          ia(i), i=1,ne
 1 3 5 7                              ja(i), i=1,n
 -1d31   10     7                     lr(i), i=1,m
 5       1d31   7                     ur(i), i=1,m
 0.0     -1     0.0                   lc(i), i=1,n
 4       1      1d31                  uc(i), i=1,n
 1       4      9                     c(i),  i=1,n
```

For either value of `type`, a starting point may be optionally supplied by providing an array of length `n` in the last row of the input file.

Problem data can also be passed directly in memory by calling the subroutine `barmilp` as shown below. The arrays should be declared as follows:

The `real*8` arrays:

$$
\begin{aligned}
c &= (1 \quad 4 \quad 9) \\
a &= (1 \quad 1 \quad 1 \quad -1 \quad 1 \quad 1) \\
lr &= (-1d31 \quad 10 \quad 7) \\
ur &= (5 \quad 1d31 \quad 7) \\
lc &= (0.0 \quad -1 \quad 0.0) \\
uc &= (4 \quad 1 \quad 1d31)
\end{aligned}
$$

The `integer*4` arrays:

$$
\begin{aligned}
ia &= (1 \quad 2 \quad 1 \quad 3 \quad 2 \quad 3) \\
ja &= (1 \quad 3 \quad 5 \quad 7)
\end{aligned}
$$

As the first two variables are integers, `numint` = 2.

The subroutine call is:

```
      call barmilp(
     $        lc, uc, n, barspace, work, optfname,
     $        barstatus, zlbest, zub, xbest, totaltime,
     $        proname, c, lr, ur, a, ia, ja, m, numint, ne)
```

The above input gives the screen output given below:

```
================================================================================
                          Welcome to BARON v. 4.0
                    Global Optimization by BRANCH-AND-REDUCE
================================================================================
                        Mixed Integer Linear Programming
================================================================================
 Preprocessing found feasible solution with value   .540000D+02
================================================================================
  Itn. no.      Open Nodes      Total Time     Lower Bound      Upper Bound
        1               1        000:00:00    -.100000D+52      .540000D+02
        1               0        000:00:00     .540000D+02      .540000D+02


                      *** Successful Termination ***


  Total time elapsed   :     000:00:00,    in seconds :           .01
       on preprocessing:     000:00:00,    in seconds :           .01
       on navigating   :     000:00:00,    in seconds :           .00
       on relaxed      :     000:00:00,    in seconds :           .00
       on local        :     000:00:00,    in seconds :           .00
       on tightening   :     000:00:00,    in seconds :           .00
       on marginals    :     000:00:00,    in seconds :           .00
       on probing      :     000:00:00,    in seconds :           .00


   Total no. of BaR iterations:       1
   Best solution found at node:       0
   Max. no. of nodes in memory:       1

 All done with problem milp
================================================================================
```

8.12 Module `barmisdp`: Mixed Integer Semidefinite Programming

$$
\begin{array}{ll}
\min & 48x_1 - 8x_2 + 20x_3 \\
\text{s.t.} & \begin{pmatrix} 10 & 4 \\ 4 & 0 \end{pmatrix} x_1 + \begin{pmatrix} 0 & 0 \\ 0 & -8 \end{pmatrix} x_2 + \begin{pmatrix} 0 & -8 \\ -8 & -2 \end{pmatrix} x_3 - \begin{pmatrix} -11 & 0 \\ 0 & 23 \end{pmatrix} \succeq \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \\
& x_1, x_2, x_3 \in \{-5, \ldots, 5\}
\end{array}
$$

The above example is a modified version of Example 1 in [6] and can be input via the following file in sparse form:

```
misdp                  problem name
 3 3 7                 numbers of: variables, integers, nonzeros
 1                     number of blocks
 2                     block structure
 48     -8     20      objective function coefficients
 0 1 1 1 -11           matrix F0, block 1, element 11
 0 1 2 2  23           matrix F0, block 1, element 22
 1 1 1 1  10           matrix F1, block 1, element 11
 1 1 1 2   4           matrix F1, block 1, element 12
 2 1 2 2  -8           matrix F2, block 1, element 22
 3 1 1 2  -8           matrix F3, block 1, element 12
 3 1 2 2  -2           matrix F3, block 1, element 22
 -5     -5     -5      lower bounds for the columns
  5      5      5      upper bounds for the columns
  0      0      0      starting point for the columns (optional)
```

In the second line, the numbers of variables, integer variables, and nonzero elements in the upper triangular part of the constraint matrices are specified. In the third line of the input file, one specifies the number of blocks of the block diagonal matrices F_i $(i = 1, ..., n)$–all have the same block pattern. The is only one two-dimensional block for this example. The dimension of each block is declared in the fourth line of the input file (a negative value denotes a diagonal block). Following the objective function coefficients, the nonzero elements of the upper diagonal part of the F-matrices are given.

Problem data can also be passed directly in memory by calling the subroutine `barmisdp` as shown below. The arrays should be declared as follows:

The `real*8` arrays:

$$\begin{aligned} c &= (\,48 \quad -8 \quad 20\,) \\ lc &= (\,-5 \quad -5 \quad -5\,) \\ uc &= (\,5 \quad 5 \quad 5\,) \\ xbest &= (\,0 \quad 0 \quad 0\,) \\ f &= (\,-11 \quad 23 \quad 10 \quad 4 \quad -8 \quad -8 \quad -2\,) \end{aligned}$$

The `integer*4` arrays:

$$\begin{aligned} bs &= (\,2\,) \\ fm &= (\,0 \quad 0 \quad 1 \quad 1 \quad 2 \quad 3 \quad 3\,) \\ fb &= (\,1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1\,) \end{aligned}$$

$$fi = (1\ \ 2\ \ 1\ \ 1\ \ 2\ \ 1\ \ 2)$$
$$fj = (1\ \ 2\ \ 1\ \ 2\ \ 2\ \ 2\ \ 2)$$

The number of integer variables for this problem is three: `numint` = 3. Finally, the number of nonzero elements in the upper triangular part of the constraint matrices is `ne` = 7.

The subroutine call is:

```
      call barmisdp(
     $        lc, uc, n, barspace, work, optfname,
     $        barstatus, zlbest, zub, xbest, totaltime,
     $        proname, numint, ne, nb, c, bs, fm, fb, fi, fj, f)
```

The above input gives the screen output given below:

```
===============================================================================
                          Welcome to BARON v. 4.0
                   Global Optimization by BRANCH-AND-REDUCE
===============================================================================
                    Mixed Integer Semidefinite Programming
===============================================================================
===============================================================================
           We have space for      666649      nodes in the tree
===============================================================================
  Itn. no.      Open Nodes      Total Time     Lower Bound      Upper Bound
        1               1        000:00:00    -.100000D+52      .100000D+52
        1               1        000:00:00    -.378618D+02      .100000D+52
*      10               1        000:00:00    -.289237D+02     -.280000D+02
       10               0        000:00:00    -.280000D+02     -.280000D+02


                     *** Successful Termination ***


  Total time elapsed   :     000:00:00,    in seconds :           .14
       on preprocessing:     000:00:00,    in seconds :           .04
       on navigating   :     000:00:00,    in seconds :           .00
       on relaxed      :     000:00:00,    in seconds :           .10
       on local        :     000:00:00,    in seconds :           .00
       on tightening   :     000:00:00,    in seconds :           .00
       on marginals    :     000:00:00,    in seconds :           .00
       on probing      :     000:00:00,    in seconds :           .00


   Total no. of BaR iterations:      10
```

```
   Best solution found at node:      10
   Max. no. of nodes in memory:       6

 All done with problem misdp
============================================================================
```

Chapter 9

Using the Parser and NLP Module

In addition to data files and direct subroutine calls, input to BARON can also be in the form of a "high level" program/file. The contents of this file are parsed and the problem is reformulated according to the module being specified in the file. This method of input is described in the current chapter.

9.1 General Usage Description

All input files to the parser should have the extension `.bar`. The file syntax is described later. The parser can be used in two different ways: to solve the problem directly or to generate the input `problem` and `options` files for BARON.

Let the input file be called `test.bar`. Then, issuing the command

```
$ barin test
```

or

```
$ barin test.bar
```

at the command line parses the file and calls BARON to solve the problem directly.

Files can also be parsed using the `-f` (*i.e.*, file) option:

```
$ barin -f test
```

or

```
$ barin -f test.bar
```

The above commands generate the `problem` and `options` files as `test.prob` and `test.opt`, respectively. These files can be used to solve the problem at a later stage. For the NLP module, the problem can only be solved directly.

9.2 Input Grammar

The following rules should be adhered to while writing an input file:

- All statements should be terminated by a semicolon (`;`).
- Reserved words must appear in uppercase letters.
- Variable names can be in lower or upper case. The parser is case sensitive, *i.e.*, `X1` and `x1` are two different variables.
- Variable names should be no longer than 15 characters.
- Variable names can be any combination of letters and numbers that starts with a letter.
- Non-alphanumeric characters such as underscores (_), hyphens (-) etc. are not permitted in variable names.
- Any text between `//` and the end of a line is ignored (*i.e.*, it is treated as a comment).
- The signs "+", "-", "*" and "/" have their usual meaning of arithmetic operations. "^" is the power/exponentiation operator where, if the base is a negative constant, the exponent must be an integer.
- The exponential function is denoted as `exp`().
- The natural logarithm is available as `log`() as well as `ln`().
- Parentheses ("(" and ")") can also be used in any meaningful combination with operations in mathematical expressions.

The input file can roughly be divided into four sections: the options, the memory assignment, the module declaration, and the problem data sections.

9.2.1 The Options Section

This section is optional. If used, it should be placed on top of the file. Any of the algorithmic options can be specified here. The module or `alg` number does not have to be specified as it is provided in the module section. The options section has the following form:

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

The `<optname>` can be any of those described in Chapter 6. `<optvalue>` is the corresponding value of that option. Options not specified here take their default values. Instead of `OPTIONS`, the word `OPTION` can also be used.

9.2.2 The Memory Section

This section is optional. If used, it should be placed after the options section and before the module section. Its purpose is to specify the memory that BARON is allowed to use. The format of the memory section is:

```
BAR_SPACE_LENGTH: <value>;
```

The effect of this command is to allocate `value` doublewords to BARON's work array. If the above command is not included in the input file, a default value of 1 million doublewords will be used for `value`.

9.2.3 The Module Section

The module section is a single statement and is required in all input files. The statement is of the form:

```
MODULE: <module_name>;
```

Here, `<module_name>` can be any one of the several described below.

Module Name	Meaning
FCP	Fixed Charge Programming
FP	Fractional Programming
GLMP	General Linear Multiplicative Programming
IQP	Indefinite Quadratic Programming
LMP	Linear Multiplicative Programming
MILP	Mixed Integer Linear Programming
NLP	Factorable Non Linear Programming
PES	Power Economies of Scale
POLY	Univariate Polynomial Programming
SCQP	Separable Concave Quadratic Programming

The module statement should be placed before the problem data in the input file.

9.2.4 The Problem Data

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

- **Variable Declaration**: All variables used in the problem have to be declared before they are used in equations. Variables can be declared as binary, integer, positive or free using the keywords BINARY_VARIABLES, INTEGER_VARIABLES, POSITIVE_VARIABLES, and VARIABLES respectively. In these keywords, VARIABLE or VAR may be used instead of VARIABLES and the underscore may be replaced by a space. Note that general integer and 0-1 variables must come first in the declaration of variables. A sample declaration is as follows:

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

  Note that **all** discrete (binary and integer) variables should be declared before **any** of the continuous variables. This is an internal requirement in BARON.

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

  A sample bounds declaration follows:

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

UPPER_BOUND{
x4: 100;
}
```

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

  A sample branching priorities section follows:

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

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

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

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

  Note that the naming rules for the equations are the same as those for variables, *i.e.*, all equation names should be lowercase and should begin with a letter.

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

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

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

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

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

  The objective function for the FCP module is given in a different manner. The contribution to the objective from each variable is listed separately. A sample FCP objective definition is:

```
OBJ: minimize FCP_FUNC {
x1: 200 - 6*x1; // 200 = fixed charge cost for x1
        //  -6 = variable cost for x1
x2: 150 - 4*x2; // 150 = fixed charge cost for x2
        //  -4 = variable cost for x2
x3: 100 - 7*x3; // 100 = fixed charge cost for x3
        //  -7 = variable cost for x3
}
```

- **Starting Point**: A starting point can be optionally specified using the keyword `STARTING_POINT`. A zero value will be used for variables whose starting point is not specified.

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

9.3 Error Messages

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

9.4 Sample Input File

A sample input file for the NLP module is shown below:

```
// This is input file ex1.bar for the NLP module
OPTIONS{
contol: 1.0e-5; // constraint tolerance
//maxiter: 4;
}
MODULE: NLP;
POSITIVE_VARIABLES x1, x2;
UPPER_BOUNDS{
x1: 6;
x2: 4;
}
EQUATION e1;
e1: x1*x2 <= 4;
OBJ: minimize -(x1 + x2);
```

Bibliography

[1] N. Adhya, M. Tawarmalani, and N. V. Sahinidis. Global Optimization of the Pooling Problem. *Industrial & Engineering Chemistry*, 38:1956–1972, 1999.

[2] A. Brooke, D. Kendrick, and A. Meeraus. *GAMS–A User's Guide*. The Scientific Press, Redwood City, CA, 1988.

[3] CPLEX. *CPLEX 6.0 User's Manual*. ILOG CPLEX Division, Incline Village, NV, 1997.

[4] M. C. Dorneich and N. V. Sahinidis. Global Optimization Algorithms for Chip Layout and Compaction. *Engineering Optimization*, 25:131–154, 1995.

[5] C. A. Floudas and P. M. Pardalos. *A Collection of Test Problems for Constrained Global Optimization Algorithms*. Springer-Verlag, Verlin, 1990.

[6] K. Fujisawa, M. Kojima, and K. Nakata. *SDPA (SemiDefinite Programming Algorithm) User's Manual — Version 5.00*. Research Reports, Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, tokyo, Japan, 1999.

[7] P. E. Gill, W. Murray, and M. A. Saunders. *User's Guide for SNOPT 5.3: A FORTRAN Package for Large-Scale Nonlinear Programming*. Technical Report, University of California, San Diego and Stanford University, CA, 1999.

[8] R. A. Gutierrez and N. V. Sahinidis. A Branch-and-bound Approach for Machine Selection in Just-in-Time Manufacturing Systems. *International J. Production Research*, 34:797–818, 1996.

[9] IBM. *Optimization Subroutine Library Guide and Reference Release 2*. International Business Machines Corporation, Kingston, NY, Third edition, 1991.

[10] M. L. Liu and N. V. Sahinidis. Process Planning in a Fuzzy Environment. *European J. Operational Research*, 100:142–169, 1997.

[11] M. L. Liu, N. V. Sahinidis, and J. P. Shectman. Planning of Chemical Process Networks via Global Concave Minimization. In I. E. Grossmann (ed.), *Global Optimization in Engineering Design,* Kluwer Academic Publishers, Boston, MA, pages 195–230, 1996.

[12] G. P. McCormick. *Converting General Nonlinear Programming Problems to Separable Nonlinear Programming Problems*. Technical Report Serial T-267, The George Washington University, Washington, D.C., 1972.

[13] G. P. McCormick. Computability of Global Solutions to Factorable Nonconvex Programs: Part I - Convex Underestimating Problems. *Mathematical Programming*, 10:147–175, 1976.

[14] G. P. McCormick. *Nonlinear Programming. Theory, Algorithms, and Applications*. Wiley Interscience, New York, 1983.

[15] B. A. Murtagh and M. A. Saunders. *MINOS 5.5 User's Guide*. Technical Report SOL 83-20R, Systems Optimization Laboratory, Department of Operations Research, Stanford University, CA, 1995.

[16] H. S. Ryoo and N. V. Sahinidis. Global Optimization of Nonconvex NLPs and MINLPs with Applications in Process Design. *Computers & Chemical Engineering*, 19:551–566, 1995.

[17] H. S. Ryoo and N. V. Sahinidis. A Branch-and-Reduce Approach to Global Optimization. *J. Global Optimization*, 8:107–139, 1996.

[18] H. S. Ryoo and N. V. Sahinidis. Analysis of Bounds for Multilinear Functions. *J. Global Optimization*, submitted, 1999.

[19] J. P. Shectman and N. V. Sahinidis. A Finite Algorithm for Global Minimization of Separable Concave Programs. *J. Global Optimization*, 12:1–36, 1998.

[20] M. Tawarmalani, S. Ahmed, and N. V. Sahinidis. Convex Extensions of l.s.c. Functions and Reformulations of Rational Functions of 0-1 Variables. *Mathematical Programming*, submitted, 1998. `http://archimedes.scs.uiuc.edu/papers/extensions.pdf`.

[21] M. Tawarmalani, S. Ahmed, and N. V. Sahinidis. Global Optimization of Fractional Programs. *J. Global Optimization*, submitted, 1999. `http://archimedes.scs.uiuc.edu/papers/fractional.pdf`.

[22] J. G. VanAntwerp, R. D. Braatz, and N. V. Sahinidis. Globally Optimal Robust Control. *J. Process Control*, 9:375–383, 1999.