Journal of Global Optimization **11:** 287–311, 1997.

A Reduced Space Branch and Bound Algorithm for Global optimization

THOMAS G. W. EPPERLY and EFSTRATIOS N. PISTIKOPOULOS
Centre for Process Systems Engineering, Imperial College of Science, Technology and Medicine, Prince Consort Road, London SW7 2BY

(Recieved: 12 March 1996; accepted: 31 January 1997)

Abstract. A general class of branch and bound algorithms for solving a wide class of nonlinear programs with branching only in a subset of the problem variables is presented. By reducing the dimension of the search space, this technique may dramatically reduce the number of iterations and time required for convergence to ϵ tolerance while retaining proven exact convergence in the infinite limit. This presentation includes specifications of the class of nonlinear programs, a statement of a class of branch and bound algorithms, a convergence proof, and motivating examples with results.

Key words: Global optimization, branch and bound, reduced space, convex envelope

1. Introduction

One line of development of branch and bound algorithms involves solving a convex relaxation of the original problem to generate the lower bound. This approach stems from the work of Falk and Soland [4], Al-Khayyal and Falk [1], and McCormick [10, 11] where convex envelopes are used to replace nonconvex terms in developing the relaxation. Recent work in this line of development has focused on developing tighter constraints in the relaxation to produce higher lower bounds [2, 3, 18, 20, 21] or on reducing the variable domain [19] with the overall goal of reducing the need to branch. Branching is avoided because it involves the combinatorial search of a multidimensional space, and the computational work can grow exponentially. The reduced space algorithm presented below is intended to complement new or existing global optimization algorithms in this line of development by reducing the dimension of the search space for a wide class of problems.

Several other researchers have presented reduced space branch and bound algorithms, but these algorithms require problems with special properties. Phillips and Rosen [16] use branch and bound in a subset of the variable space in solving minimizations of a concave quadratic function over a bounded polyhedral set. Their algorithm branches only on the nonlinear variables, and it is able to solve problems with numerous linear variables. Horst and Thoai [7] presented a conical branch and bound algorithm for solving concave problems with numerous variables appearing linearly. The dimension of the problem to be solved via the branch and bound algorithm is only one dimension greater than the number of nonlinear vari-

ables. Muu and Oettli [13, 14] present a combined branch and bound and cutting plane algorithm to perform branch and bound in a subset of the problem variables. Their algorithm can treat problems with convex-concave constraints. Sherali and Alameddine [20] present a branch and bound algorithm based on the reformulation-linearization technique which has proven convergence for bilinear problems when branching on a subset of the variables. They point out that the algorithm can branch on the smaller of the two sets of variables. Horst and Thoai [8] present an algorithm for d.c. programming problems that branches in a reduced space. Phong, An, and Tao [17] present a reduced space branch and bound algorithm for indefinite quadratic programs which branches only on the concave variables.

The reduced space algorithm presented below is applicable to sparse nonlinear programs that can be expressed in factorable form, which includes most practical problems [11]. Smith and Pantelides [23] present an algorithm for reformulating expressions into the necessary form. Not all programs in factorable form will be able to branch in a reduced space, but any problem in factorable form can be quickly and automatically tested to see if it can use a reduced space.

The class of problems that can be solved in a reduced space is detailed in Section 1. Section 2 details the convex constraints that must be included in the convex relaxation used to obtain the lower bound. The reduced space algorithm is given in Section 3, and a convergence proof is provided in Section 4. Section 5 includes a comparison of full and reduced space solutions of three example problems, and Section 6 has some brief concluding remarks. The appendix contains a proposed algorithm for determining the minimal set of branching variables and details of the example problems.

2. Problem Definition

Consider the problem,

Problem (P1)

$$\begin{aligned} \min_{x,y} \quad & f(x,y) && (1)\\ \text{s.t.} \quad & g(x,y) \leq 0 && (2)\\ & h(x,y) \leq 0 && (3)\\ & x \in X && (4)\\ & y \in Y && (5) \end{aligned}$$

with

$$X := [\underline{x}, \bar{x}] \subset R^N, Y := [\underline{y}, \bar{y}] \subset R^M$$
$$f : X \times Y \to R, g : X \times Y \to R^G, h : X \times Y \to R^H.$$

Equality constraints may be included in this framework as a pair of opposing inequality constraints. In this formulation, the y variables require branching, and

the x variables do not. A proposed algorithm to determine X and Y for a program in factorable form is included in the appendix. To facilitate the presentation below, D is introduced as the feasible space of problem P1,

$$D := \{(x, y) \in X \times Y : g(x, y) \leq 0, h(x, y) \leq 0\}. \tag{6}$$

In order to guarantee convergence of the branch and bound algorithm while branching only in the Y space, it is necessary to place some restrictions on the functions f, g, and h. g must be a convex function, and f and each element of h must satisfy the conditions given below for function w. Let w be any element of h or f, so $w : X \times Y \to R$. Convergence can be established when w has the following form:

$$w(x, y) := w^A(x) + \sum_{i \in Q} w_i^B(x) w_i^C(y) + w^D(y) \tag{7}$$

where Q is a set of indices indicating the bilinear interactions between functions of x and functions of y. w^A and w^B must be convex, and w^C and w^D must be continuous. Further conditions on w^A, w^B, w^C, and w^D are stated below after the necessary terminology has been defined.

It is assumed that $\underline{x}$, $\bar{x}$, $\underline{y}$, and $\bar{y}$ are finite; thus, X and Y are compact sets that can be thought of as N and M dimensional rectangles. The partition sets used by this algorithm are all rectangular and compact with

$$X^k := [\underline{x}^k, \bar{x}^k], \tag{8}$$

$$Y^k := [\underline{y}^k, \bar{y}^k]. \tag{9}$$

Tuy [24] showed that rectangular subdivisions are weakly exhaustive. Some definitions are required for the presentation of the underestimating program, algorithm and proof.

DEFINITION 1. Given a set $X \times Y$, the subdivision of partition sets of $X \times Y$ is called exhaustive on Y, if it produces an infinite, nested sequence of partition sets, $\{(X^k, Y^k)\}$ and an associated sequence of diameters $d(Y^k)$ satisfying the following:

$$(X^0, Y^0) \supset (X^1, Y^1) \supset \cdots \supset (X^k, Y^k) \tag{10}$$

$$\lim_{k \to \infty} d(Y^k) = 0 \tag{11}$$

$$\lim_{k \to \infty} Y^k = \bigcap_k Y^k = \{\dot{y}\}. \tag{12}$$

This definition is an extension of Horst's [6] whose definition is recovered when X is empty in which case the subdivision is just called exhaustive.

DEFINITION 2. A convex underestimator of a function $\varphi(y)$ over the domain Y^k is a convex function denoted $\breve{\varphi}^k(y)$ satisfying

$$\breve{\varphi}^k(y) \leq \varphi(y) \quad \forall y \in Y^k. \tag{13}$$

DEFINITION 3. A concave overestimator of a function $\varphi(y)$ over the domain Y^k is a concave function denoted $\hat{\varphi}^k(y)$ satisfying

$$\hat{\varphi}^k(y) \geq \varphi(y) \quad \forall y \in Y^k. \tag{14}$$

DEFINITION 4. Given a sequence of partition sets $\{(X^k, Y^k)\}$ produced by a subdivision of $X \times Y$ that is exhaustive on Y with $Y^k \underset{k\to\infty}{\longrightarrow} \{\dot{y}\}$, a corresponding sequence $\{y^k\}$ with $y^k \in Y^k$, and a corresponding sequence $\{x^k\}$ with $x^k \in X^k$ with $\{x^k\} \to \dot{x}$, a convex underestimator or concave overestimator is strongly consistent on Y if there exist subsequences $\{(X^q, Y^q)\}$, $\{y^q\}$, and $\{x^q\}$ satisfying

$$\breve{\varphi}^q(x^q, y^q) \underset{q\to\infty}{\longrightarrow} \varphi(\dot{x}, \dot{y}) \quad \text{or} \tag{15}$$

$$\hat{\varphi}^q(x^q, y^q) \underset{q\to\infty}{\longrightarrow} \varphi(\dot{x}, \dot{y}). \tag{16}$$

When X is vacuous, a convex underestimator or concave overestimator is called simply strongly consistent.

DEFINITION 5. A function, φ, has tight bounds if one can provide upper and lower bounds, $\bar{\varphi}^k$ and $\underline{\varphi}^k$ respectively, for any subrectangle Y^k of Y such that for any infinite sequence, $\{Y^k\}$, produced by an exhaustive subdivision with $Y^K \to \{\dot{y}\}$,

$$\underline{\varphi}^k \leq \varphi(y) \leq \bar{\varphi}^k \quad \forall y \in Y^k, \tag{17}$$

$$\lim_{k\to\infty} \underline{\varphi}^k = \lim_{k\to\infty} \bar{\varphi}^k = \varphi(\dot{y}). \tag{18}$$

A function, φ has continuous tight bounds if $\bar{\varphi}^k$ and $\underline{\varphi}^k$ are continuous functions of $\underline{y}^k$ and $\bar{y}^k$.

Having defined these concepts and notation, the sufficient criteria for convergence of the algorithm is that f and h have convex underestimators that are strongly consistent on Y. Such convex underestimators are available when w satisfies the following formal conditions:

Conditions W

1. w^A, w^B are convex
2. w^C, w^D are continuous
3. strongly consistent, convex underestimators are available for w^C and w^D
4. strongly consistent, concave overestimators are available for w^C and w^D
5. w^B, w^C, w^D have continuous tight bounds
6. For each $i \in Q$, at least one of the following two conditions must hold
   (a) $w_i^B(x) := c^{i\mathrm{T}}x$ for some constant $c^i \in R^N$
   (b) $w_i^C(y) \geq 0$ for all $y \in Y$; hence, $\underline{w}_i^{Ck} \geq 0$

The following examples demonstrate some of the breadth of this class of functions:

$$
\begin{aligned}
w(x,y) &:= x^{\mathrm{T}}x + y^{\mathrm{T}}By,\\
w(x,y) &:= x^{\mathrm{T}}x + x^{\mathrm{T}}Ay + y^{\mathrm{T}}By,\\
w(x,y) &:= c^{\mathrm{T}}x + \sum_i x_i^2 \exp(y_i).
\end{aligned}
$$

Given convex underestimators for f and h, the following convex nonlinear program is used to provide lower bounds and to test the feasibility of a partition set.

Problem (P2)

$$
\begin{aligned}
\min_{x,y} \quad & \breve{f}^k(x,y) && (19)\\
\text{s.t.} \quad & g(x,y) \le 0 && (20)\\
& \breve{h}^k(x,y) \le 0 && (21)\\
& x \in X^k && (22)\\
& y \in Y^k. && (23)
\end{aligned}
$$

Based on the definition of convex underestimators, every feasible point of P1 in the subdomain $X^k \times Y^k$ is feasible in P2; and the objective of P2 is less than or equal to that of P1 for all points in $X^k \times Y^k$. Thus, P2 provides a valid lower bound for the solution of P1 over the partition set X^k and Y^k. It should be noted that problem P2 contains only the necessary constraints to guarantee convergence of the algorithm. If available, one may add tighter bounding constraints to P2 such as [2, 3, 18] as long as the constraints given in P2 remain.

3. Convex Underestimators

In this section, convex underestimators for f and h satisfying conditions W will be provided and proven to be strongly consistent on Y. This proof establishes the sufficient conditions for the underestimating program to guarantee convergence of the algorithm given below for a wide class of problems.

A convex underestimator of (7) is:

$$
\breve{w}^k(x,y) := w^A(x) + \breve{w}^{Dk}(y) + \sum_{i\in Q} W_i^k(x,y) \qquad (24)
$$

with

$$
W_i^k(x,y) :=
$$

$$\begin{cases} \max\left\{ \begin{array}{l} \bar{w}_i^{Bk}\breve{w}_i^{Ck}(y) + w_i^B(x)\bar{w}_i^{Ck} - \bar{w}_i^{Bk}\bar{w}_i^{Ck}, \\ \underline{w}_i^{Bk}\breve{w}_i^{Ck}(y) + w_i^B(x)\underline{w}_i^{Ck} - \underline{w}_i^{Bk}\underline{w}_i^{Ck} \end{array} \right\} & \text{if } \underline{w}^{Bk} \geq 0 \\ \max\left\{ \begin{array}{l} \bar{w}_i^{Bk}\hat{w}_i^{Ck}(y) + w_i^B(x)\bar{w}_i^{Ck} - \bar{w}_i^{Bk}\bar{w}_i^{Ck}, \\ \underline{w}_i^{Bk}\hat{w}_i^{Ck}(y) + w_i^B(x)\underline{w}_i^{Ck} - \underline{w}_i^{Bk}\underline{w}_i^{Ck} \end{array} \right\} & \text{if } \bar{w}^{Bk} < 0 \\ \max\left\{ \begin{array}{l} \bar{w}_i^{Bk}\breve{w}_i^{Ck}(y) + w_i^B(x)\bar{w}_i^{Ck} - \bar{w}_i^{Bk}\bar{w}_i^{Ck}, \\ \underline{w}_i^{Bk}\hat{w}_i^{Ck}(y) + w_i^B(x)\underline{w}_i^{Ck} - \underline{w}_i^{Bk}\underline{w}_i^{Ck} \end{array} \right\} & \text{otherwise} \end{cases} \tag{25}$$

The proof that (24) is a convex underestimator of w follows from [10, 11], and these references also provide a method for providing the strongly consistent, convex underestimators and concave overestimators for the nonconvex functions of y. The convexity of (25) follows from the definitions of convex underestimators and convex overestimators, property 6, and the fact that a maximum of two convex functions is convex. Each term of (24) is convex, so (24) is convex.

LEMMA 1. *Given a function, w, in form (7) satisfying properties 1–6, then $\breve{w}$ given by (24) is strongly consistent on Y.*

Proof. The definition provides that there is a sequence of subrectangles $\{(X^k, Y^k)\}$ produced by a subdivision that is exhaustive on Y, a corresponding sequence $\{y^k\}$ with $y^k \in Y^k$, and a corresponding sequence $\{x^k\}$ with $x^k \in X^k$ and $x^k \to \dot{x}$. From the definition of an exhaustive subdivision, we have that $Y^k \to \{\dot{y}\}$, and it follows that $y^k \to \dot{y}$. The upper and lower bounds on X^k are both sequences in a compact space, so there exists a convergent subsequence, so $X^q \to X^* = [\underline{x}^*, \bar{x}^*]$. Corresponding to this subsequence, there is the sequence $\{(\underline{x}^q, \bar{x}^q, \underline{y}^q, \bar{y}^q, y^q, x^q)\} \to (\underline{x}^*, \bar{x}^*, \dot{y}, \dot{y}, \dot{y}, \dot{x})$. From properties 1–5, $\breve{w}^q$ can be considered as a continuous function of $(\underline{x}^q, \bar{x}^q, \underline{y}^q, \bar{y}^q, y^q, x^q)$; therefore, the limit of $\breve{w}^q$ as $q \to \infty$ is just $\breve{w}^q$ evaluated at the limiting values.

By property 3

$$\lim_{q\to\infty} \breve{w}^q(x^q, y^q) = w^A(\dot{x}) + w^D(\dot{y}) + \sum_{i\in Q} W_i^\infty(\dot{x}, \dot{y}),$$

and applying properties 3 and 4 to W_i^∞, equation (25) above with the limiting values inserted.

$$W_i^\infty(\dot{x}, \dot{y}) = w_i^B(\dot{x}) w_i^C(\dot{y})$$

which completes the requirements for a strongly consistent convex underestimator. □

4. Algorithm

This description of a general class of branch and bound algorithms is based largely on the ideas of previous researchers [6, 11, 18, 19]. Here we present a general purpose branch and bound algorithm that includes branching on a subset of the variable

domain explicitly. Steps marked as optional can be left out without effecting the convergence proof that follows; in practice, they are helpful for some problems. Let $\beta(M^k)$ or $\beta(X^k, Y^k)$ refer to the optimal objective function value of P2 for the region $M^k = (X^k, Y^k)$ and $z(M^k)$ or $z(X^k, Y^k)$ refer to an element of the corresponding argmin.

ALGORITHM 1. Reduced space branch and bound algorithm

Initialization (iteration 0)

Step 1. Apply any finite methods to reduce the initial variable domain size without removing a global optimum (e.g. see the preprocessing step [18]) to produce $X^0 \subseteq X$ and $Y^0 \subseteq Y$. (optional)

Step 2. Solve problem P2, $\beta_0 := \beta(X^0, Y^0)$, and $z^0 := z(X^0, Y^0)$. If problem P2 is infeasible, stop; problem P1 is infeasible.

Step 3. Apply finite variable domain reduction techniques guaranteed not to remove a global optimum [19, 22] to produce $X^{0\prime} \subseteq X^0$ and $Y^{0\prime} \subseteq Y^0$. If $X^{0\prime} \neq X^0$ or $Y^{0\prime} \neq Y^0$ and the limit to the number of variable domain reductions has not run out, $X^0 := X^{0\prime}$ and $Y^0 := Y^{0\prime}$ repeat step 3; otherwise, $X^0 := X^{0\prime}$ and $Y^0 := Y^{0\prime}$. (optional)

Step 4. Initialize the iteration counter $k := 1$, the partition $\underline{M}_0 := \{(X^0, Y^0)\}$, the upper bound $\alpha_0 := \infty$, and the set of feasible points $F^0 := \emptyset$.

Main Loop (iterations $1, 2, \ldots$)

Step 5. Apply some finite method to search for feasible points (potential optima) of P1 such as applying a local NLP solver to P1. Let F be the potentially empty set of feasible points located by this procedure, assign $F^k := F \cup F^{k-1}$. This step need not be applied at every iteration of the main loop once a feasible point has been found. If it is skipped, $F^k := F^{k-1}$.

$$\alpha_k := \min_{z \in F^k} f(z) \tag{26}$$

If $F^k \neq \emptyset$, define the best known feasible point

$$b^k := \operatorname*{argmin}_{z \in F^k} f(z). \tag{27}$$

Step 6. Remove elements of the current partition that cannot contain a solution

$$\underline{R}_k := \{M \in \underline{M}_{k-1} : \beta(M) < \alpha_k\}.$$

Step 7. Choose a nonempty collection $P_k \subseteq \underline{R}_k$, and partition each element of P_k into subrectangles *only branching in the Y space.* Call the new partition sets P'_k.

Step 8. For each $P \in P'_k$ perform the following steps:

1. Apply finite variable domain reduction techniques guaranteed not to remove a global optimum [19] to produce $P^* \subseteq P$. Replace $P \in P'_k$ with P^*, and $P := P^*$. (optional)
2. Solve P2 to obtain $\beta(P)$, if P2 is infeasible or $\beta(P) > \alpha_k$ remove P from P'_k and skip to next element of P'_k.
3. Apply finite variable domain reduction techniques guaranteed not to remove a global optimum [19] to produce $P^* \subseteq P$. Replace $P \in P'_k$ with P^*. (optional)
4. If the number of variable domain reductions allowed has not run out and $P^* \neq P$, set $P := P^*$ and return to step 2. (optional)

Step 9. The partition set remaining is now

$$\underline{M}_k := (\underline{R}_k \setminus P_k) \cup P'_k \tag{28}$$

giving a new lower bound of

$$\beta_k := \inf_{M \in \underline{M}_k} \beta(M). \tag{29}$$

If $\underline{M}_k$ is empty, $\beta_k = \infty$. For the sake of the convergence proof, if $\underline{M}_k \neq \emptyset$ the following definitions apply

$$M_k \in \operatorname*{argmin}_{M \in \underline{M}_k} \beta(M), \tag{30}$$

$$z^k := z(M_k). \tag{31}$$

Step 10. If $\alpha_k - \beta_k > 0 (\geq \epsilon)$, $k := k + 1$ and return to step 5; otherwise, the problem is solved. If $F^k = \emptyset$, the problem is infeasible; otherwise, $\alpha_k = \beta_k$ is the solution of P1, and b^k is an optimal solution.

This algorithm includes the possibility of reducing the domain of the x variables, but it is not required for convergence. If the variable domain reduction techniques are not used, $X^0 = X^1 = X^2 \cdots$.

5. Convergence Proof

Let Z^a be the set of accumulation points of z^k, and let Z^* be the $\operatorname{argmin}_{(x,y)\in D} f(x, y)$.

THEOREM 1. *If a reduced space branch and bound algorithm satisfying the following conditions*

1. *the subdivision of partition sets in step 7 is exhaustive on Y*
2. *the selection of elements to be partitioned in step 7 is bound improving*
3. *the convex subfunctionals, $\breve{f}$ and $\breve{h}$, used in problem P2 are strongly consistent on Y*

is applied to problems in the form P1 with $D \neq \emptyset$, then an infinite application of the algorithm will produce

1. $\beta := \lim_{k\to\infty} \beta_k = \min_{(x,y)\in D} f(x,y)$
2. $Z^a \subset Z^*$

Proof. For every iteration, $k = 0, 1, 2, \ldots$, of the algorithm, by design the following is true

$$\beta_k \leq \min_{(x,y)\in D} f(x,y), \tag{32}$$

$$M_k \in \operatorname*{argmin}_{M\in\underline{M}_k} \beta(M), \tag{33}$$

$$(x^k, y^k) = z^k \in z(M_k). \tag{34}$$

Horst [6] gives that $\{\beta_k\}$ is a nondecreasing sequence bounded above by $\min_{(x,y)\in D} f(x,y)$, which guarantees the existence of the limit

$$\beta := \lim_{k\to\infty} \beta_k \leq \min_{(x,y)\in D} f(x,y). \tag{35}$$

$\{z^k\}$ is a sequence on a compact set, therefore, it has a convergent subsequence. For any $(\check{x}, \check{y}) = \check{z} \in Z^a$, there exists a subsequence $\{z^r\}$ of $\{z^k\}$ with

$$\lim_{r\to\infty} z^r = \check{z}. \tag{36}$$

From properties 1 and 2, we have based on previous work [5], there exists a decreasing subsequence $Y_q \subset Y_r$ where Y_r is the Y space of the partition M_r with

$$y^q \in Y_q, \tag{37}$$

$$(x^q, y^q) \in M_q, \tag{38}$$

$$\beta_q = \beta(M_q) = \check{f}^q(x^q, y^q), \tag{39}$$

$$\lim_{q\to\infty} y^q = \{\check{y}\}. \tag{40}$$

By property 3, it follows that

$$\lim_{q\to\infty} \beta_q = \beta = f(\check{x}, \check{y}). \tag{41}$$

All that remains is to prove that $(\check{x}, \check{y}) \in D$. X and Y are closed sets, so $(\check{x}, \check{y}) \in X \times Y$. The remainder of the proof will be by contradiction.

Assume $(\check{x}, \check{y}) \notin D$, then

$$\max\{\max_i g_i(\check{x}, \check{y}), \max_j h_j(\check{x}, \check{y})\} = \delta > 0. \tag{42}$$

There are two cases, either $g_i(\check{x}, \check{y}) = \delta > 0$ for some i or $h_j(\check{x}, \check{y}) = \delta > 0$ for some j. Consider the first case, g_i is convex and hence continuous; therefore, the sequence $\{g_i(x^q, y^q)\}$ converges to $g_i(\check{x}, \check{y})$. By definition of convergence,

$\exists\ q_\delta$ such that $q > q_\delta \rightarrow |g_i(x^q, y^q) - g_i(\check{x}, \check{y})| < \delta$. Therefore for $q > q_\delta$, $g_i(x^q, y^q) > 0$ implying that P2 is infeasible and violating the assumption that $(x^q, y^q) = z^q = z(M_q)$.

The second case is that $h_j(\check{x}, \check{y}) = \delta > 0$ for some j. From the property 3, the sequence $\{\check{h}_j^q(x^q, y^q)\}$ converges to $h_j(\check{x}, \check{y})$. By a similar argument, $\exists\ q_\delta$ such that $q > q_\delta \rightarrow \check{h}_j^q(x^q, y^q) > 0$ implying that P2 is infeasible. Both cases result in a contradiction, therefore $(\check{x}, \check{y}) \in D$.

$$\beta = f(\check{x}, \check{y}) = \min_{(x,y)\in D} f(x, y) \tag{43}$$

$$(\check{x}, \check{y}) \in Z^* \tag{44}$$

□

6. Examples

We have chosen three examples to illustrate the breadth of problems to which the reduced space algorithm may be applied and to demonstrate how branching in the reduced space can dramatically decrease the runtime and number of iterations required to converge to a given tolerance. In each example, the dimension of y is much less than the dimension of x, which is the case where this approach is most effective. Before the examples and results are presented, we provide a description of our test program.

6.1. Implementation details

Branching in the reduced space can be used with any algorithm satisfying the general description provided above. For the sake of testing, we implemented a branch and bound algorithm in C ++ on a 50MHz Sun SPARC 10. The program reads in a problem description file, automatically determines the minimal set of variables that require branching using the proposed algorithm in the appendix, solves the problem, and outputs the results. When testing with full space branching, the second step is skipped; all other aspects of the algorithm are kept constant between the two modes.

There are many important details that go into the implementation of an algorithm of the form given above. Only the most crucial aspects of the algorithm will be described here. Optional steps 1, 3 and 8.1 are omitted, and the number of variable domain reductions per main loop iteration in step 8.3 is limited to four.

The LP and/or NLP subproblems generated by the program in steps 5 and 8.2 use the convex/concave envelopes as presented by McCormick [11] to provide the lower bounding relaxations, and they are solved with direct calls to MINOS 5.5 [12] using warm or hot starts when possible. In step 5, MINOS 5.5 is applied to the original problem P1 to search for local minima. The program solves P1 each main loop iteration until it finds a local minimum, and afterwards, it solves P1 with

decreasing frequency. The underestimating program solved in step 8.2 is equivalent to P2; no additional constraints are used. In step 8.3, active variable bounds are used to reduce the variable domain according to the methods of Ryoo and Sahinidis [19].

The last crucial detail of the implementation is the splitting strategy employed. In step 7, the region with the least lower bound is split into two subrectangles. The splitting process works in two steps: choosing which variable to split on and choosing where to split. Before entering step 7, the program solved the relaxation P2 for the region to be split. The relaxation provides a point and estimates of the nonconvex terms. The variable is chosen to maximize the difference between the estimate of a nonconvex term and the actual value of the nonconvex term at the relaxation solution. The choice of variable is restricted to the reduced space unless the full space option is requested.

Assume that i is the index of the chosen variable. If F^k, the set of known feasible points, is not empty, b^k is the best known feasible solution. y^k is the solution of the relaxed problem if available. Here is the algorithm used to determine the split location:

ALGORITHM 2. Method for choosing a split location

IF ($F^k \neq \emptyset$) THEN
  IF ($b_i^k \in$ interior($[\underline{y}_i^k + \epsilon^{0.6}, \bar{y}_i^k - \epsilon^{0.6}]$)) THEN Split at b_i^k
  ELSE
    lowerCloser := $|b_i^k - \underline{y}_i^k| \leq |b_i^k - \bar{y}_i^k|$
    IF (No Relaxed Solution) THEN
      IF (lowerCloser) THEN Split at $0.1\bar{y}_i^k + 0.9\underline{y}_i^k$
      ELSE Split at $0.9\bar{y}_i^k + 0.1\underline{y}_i^k$
    ELSE
      IF (lowerCloser) THEN
        $y_i^k := \max\{y_i^k, \underline{y}_i^k + 0.01(\bar{y}_i^k - \underline{y}_i^k)\}$
        $y_i^k := \min\{y_i^k, \bar{y}_i^k - 0.05(\bar{y}_i^k - \underline{y}_i^k)\}$
      ELSE
        $y_i^k := \max\{y_i^k, \underline{y}_i^k + 0.05(\bar{y}_i^k - \underline{y}_i^k)\}$
        $y_i^k := \min\{y_i^k, \bar{y}_i^k - 0.01(\bar{y}_i^k - \underline{y}_i^k)\}$
      END
      Split at y_i^k
    END
  END
ELSE
  IF (No Relaxed Solution) THEN
    Split at $(\underline{y}_i^k + \bar{y}_i^k)/2$
  ELSE
    $y_i^k := \max\{y_i^k, \underline{y}_i^k + 0.05(\bar{y}_i^k - \underline{y}_i^k)\}$

$y_i^k := \min\{y_i^k, \bar{y}_i^k - 0.05(\bar{y}_i^k - \underline{y}_i^k)\}$
Split at y_i^k
END
END

ϵ is small positive number which indicates machine precision. This splitting procedure guarantees that the width of the intervals produced are bounded above by either $(\bar{y}_i^k - \underline{y}_i^k - \epsilon^{0.6})$ (case i) or by $0.99(\bar{y}_i^k - \underline{y}_i^k)$ (case ii). In an infinite application of branching procedure to variable i, case i can only apply a finite number of times. Let j be the index of the last time that case i applied. For all remaining elements in the infinite sequence, $0 \leq \bar{y}_i^l - \underline{y}_i^l \leq 0.99^{l-j}(\bar{y}_i^j - \underline{y}_i^j)$. $\lim_{l\to\infty}(\bar{y}_i^l - \underline{y}_i^l) = 0$.

A relative tolerance of 0.0001 was used as the termination criteria for all the examples. The algorithm also terminates if a time limit of 100000 CPU seconds or an iteration of 1000000 is exceeded. The first two examples were solved to tolerance using both full and reduced space modes, and the last example was solved to tolerance only in the reduced space mode.

6.2. BILINEAR EXAMPLE

This randomly generated problem has a linear objective function, ten linear constraints and twenty bilinear constraints. It has twenty-three variables of which only three need branching with the reduced space algorithm.

$$\min_{x,y} \quad c^{\mathrm{T}} \begin{bmatrix} x \\ y \end{bmatrix} \tag{45}$$

$$\text{s.t.} \quad A \begin{bmatrix} x \\ y \end{bmatrix} \leq b \tag{46}$$

$$\sum_{i=1}^{20} \sum_{j=1}^{3} B_{kij} x_i y_j \leq d_k \; ; \; k = 1, \ldots, 20 \tag{47}$$

$$-2 \leq y_j \leq 2 \; ; \; j = 1, \ldots, 3 \tag{48}$$

$$-2 \leq x_i \leq 2 \; ; \; i = 1, \ldots, 20 \tag{49}$$

with

$$c \in R^{23}$$
$$A \in R^{10\times 23}$$
$$B \in R^{20\times 20\times 3}$$

Tables IV, V, VI, and VII, appearing in the appendix, provide the nonzero elements of c, d, A, and B respectively.

The algorithm statistics comparing the reduced space and full space modes are shown in Table I. Both versions of the algorithm locate the same global minimum

in an acceptable amount of time; however, the reduced space version uses much fewer iterations and much less CPU time. In the process of solving this problem, three local minima were located by MINOS of which two are not global minima. The objective function values of these local minima are -15.2843, -17.6430, and -20.8008. The number of relaxed problems is higher than the number of branch and bound iterations because the relaxed problem is resolved with a hot start after a variable domain reduction. The full space algorithm requires approximately eight times the CPU time and five times the number of iterations required by the reduced space algorithm.

Table I. Algorithm statistics for example one

Space	Iterations	CPU time (s)	Relaxations solved	Original NLPs solved
Reduced	995	184.9	1472	34
Full	5157	1415.3	5961	79

6.3. PRODUCTS OF CONVEX AND LINEAR FUNCTIONS

The second example involves minimizing a convex objective function subject to constraints involving products of convex and linear functions. It is a small and simplified version of the formulation for batch reactor design under uncertainty developed by Ierapetritou and Pistikopoulos [9]. The problem has thirty-seven variables of which only two need branching with the reduced space algorithm. There are six linear constraints and sixteen nonconvex constraints. The problem has at least one non-global local minimum.

$$\min_{x,y} \quad \sum_{j=1}^{3} 3\exp(0.6x_j) + c^{\mathrm{T}} \begin{bmatrix} x \\ y \end{bmatrix} \tag{50}$$

$$\text{s.t.} \quad x_j - y_i \geq \log(S_{ij}) \; ; \; i = \{1,2\}, j = \{1,2,3\} \tag{51}$$

$$\sum_{i=1}^{2} x_{(3+2j+i)} \exp(t_i - y_i) \leq 8 \; ; \; j = \{0,\ldots,15\} \tag{52}$$

$$\underline{x} \leq x \leq \bar{x} \tag{53}$$

$$\underline{y} \leq y \leq \bar{y} \tag{54}$$

with

$$S = \begin{bmatrix} 2 & 3 & 4 \\ 4 & 6 & 3 \end{bmatrix} \tag{55}$$

$$t = \begin{bmatrix} 2.995732273553991 \\ 2.772588722239781 \end{bmatrix} \tag{56}$$

The variable bounds and objective function coefficients are presented in Table VIII.

Both versions find the same solution to this example, but the difference in time required for solution between the two versions is larger than example one. Table II shows the statistics for the two algorithms. The reduced space algorithm is faster than the full space algorithm by two orders of magnitude in CPU time, and it requires three orders of magnitude less iterations than the full space algorithm.

Table II. Algorithm statistics for example two

Space	Iterations	CPU time (s)	Relaxations solved	Original NLPs solved
Reduced	25	9.87	31	5
Full	30789	6489.23	54174	176

6.4. PRODUCTS OF NONCONVEX AND CONVEX FUNCTIONS

The third example problem has a linear objective function, one linear constraint, and one nonconvex constraint involving the products of nonconvex, cubic functions and convex quadratic functions. The objective function coefficients for x were randomly determined, and the objective function coefficients for y were chosen. The constraints were chosen to give a nontrivial problem involving products of nonconvex and convex functions. The problem has twenty-three variables of which three require branching in the reduced space algorithm, and it has at least two non-global local minima.

$$\min_{x,y} \quad c^{\mathrm{T}} \begin{bmatrix} y \\ x \end{bmatrix} \tag{57}$$

$$\text{s.t.} \quad \sum_{i=1}^{8} (0.1y_1^3 - 0.2y_1^2 + 0.01y_1 + 10)0.5x_i^2 + \sum_{i=9}^{16} (0.1y_2^3 - 0.2y_2^2 + 0.01y_2 + 10)0.5x_i^2 + \sum_{i=16}^{20} (0.1y_3^3 - 0.2y_3^2 + 0.01y_3 + 10)0.5x_i^2 \leq 250 \tag{58}$$

$$y_1 + y_2 + y_3 = 0.5 \tag{59}$$

$$-2 \leq y_i \leq 2 \; ; \; i = \{1, 2, 3\} \tag{60}$$

$$0 \leq x_i \leq 10 \; ; \; i = \{1, \ldots, 20\} \tag{61}$$

The objective function coefficients appear in Table IX contained in the appendix.

The reduced space algorithm was able to find the global minimum to the specified tolerance within an acceptable amount of time, but the full space algorithm

exceeded the CPU time limit of 100000 CPU seconds. Table III shows the iterations required for the reduced space algorithm to converge to the specified tolerance, and for the full space algorithm, it shows the tolerance achieved within the CPU time limit. The reduced space algorithm requires only 4.664 CPU minutes, and the full space algorithm used over 27.77 CPU hours without converging.

Table III. Algorithm statistics for example three

Space	Iterations	CPU time (s)	Relaxations solved	Original NLPs solved	Tolerance
Reduced	433	280	529	21	1.0×10^{-4}
Full	127328	100000	130730	462	2.3×10^{-3}

7. Conclusions

These examples demonstrate that the reduced space branch and bound algorithm can provide a dramatic reduction in the CPU time required to solve problems to finite tolerance when $N \gg M$, and the proofs given provide for exact convergence in the infinite limit. The examples also demonstrate some of the breadth of problems to which this technique can be applied.

Branching in a reduced space can be incorporated easily into existing branch and bound algorithms. The variables can be partitioned automatically or manually into subsets x and y, and existing programs can use them by forbidding branches on variables in x.

Appendix

A. Automatic partitioning of variables

In this appendix, a proposed algorithm is presented to automatically determine the minimum set of variables, Y, that require branching for problems presented in the following factorable form:

$$z_j = x_j \; ; \; j = 1, \ldots, n \tag{62}$$

$$z_j = \sum_{i=1}^{j-1} \left(c_i^j z_i + \xi_i^j(z_i) + \sum_{k=1}^{j-1} B_{ik}^j z_i z_k \right) \; ; \; j = n+1, \ldots, N \tag{63}$$

Some of the elements of z are just intermediate variables added to put the problem in factorable form, but others are function values that are constrained by some lower or upper bound, z^L or z^U respectively. If z_j is an intermediate value, $z_j^L = -\infty$ and $z_j^U = \infty$; otherwise, at least one of the bounds is finite. In a sparse problem, most of the elements of each c^j and B^j are zero, and many of the functions $\xi_i^j(\cdot)$ are functions whose value is always zero.

Let I_j be the set of variables incident on the expression for z_j.

$$I_j := \{i \in 1, \ldots, j-1 : c_i^j \neq 0 | \xi_i^j(\cdot) \neq 0 | \exists k \text{ with } (B_{ik}^j \neq 0 | B_{ki}^j \neq 0)\}$$

It is also useful to define the BaseVars(j) function that determines the set of original variables on which the value of z_j depends.

$$\text{BaseVars}(j) := \begin{cases} \{x_j\} & \text{for } j \leq n \\ \bigcup_{i \in I_j} \text{BaseVars}(i) & \text{otherwise} \end{cases} \tag{64}$$

The assumptions of this work provide that all original variables have finite lower and upper bounds, $\underline{x}$ and $\bar{x}$ respectively. Lower and upper bounds for z, $\underline{z}$ and $\bar{z}$ respectively, are determined using existing interval mathematics techniques [15]. In addition to being able to provide bounds for z, it must be possible to determine whether each of the single variable functions ξ is convex, concave, both, or neither over a specified domain. Let the following functions return a boolean value indicating whether the function has the indicated curvature over the domain $[\underline{z}_i, \bar{z}_i]$.

$$\text{Concave}(\xi_i^j, \underline{z}_i, \bar{z}_i) := \begin{cases} \text{TRUE} & \text{if } \xi_i^j \text{ is concave on } [\underline{z}_i, \bar{z}_i] \\ \text{FALSE} & \text{otherwise} \end{cases} \tag{65}$$

$$\text{Convex}(\xi_i^j, \underline{z}_i, \bar{z}_i) := \begin{cases} \text{TRUE} & \text{if } \xi_i^j \text{ is convex on } [\underline{z}_i, \bar{z}_i] \\ \text{FALSE} & \text{otherwise} \end{cases} \tag{66}$$

The algorithm is presented as three subroutines and a main driver. The purpose of the first two subroutines is to check a particular line of the formulation for the indicated curvature and to add variables to Y to guarantee that the specified envelope becomes tight in the infinite limit. These subroutines also accumulate the set of bilinear interactions in A. Each element of A is an ordered pair of nonempty sets (L, R) where L is the set of variables appearing on the left hand side of a multiplication and R is the set of variables appearing on the right hand side. The third subroutine is used to choose which side of a bilinearity should be added to Y. These subroutines are directed by the main driver.

ALGORITHM 3. Subroutine to ensure a tight convex envelope

This subroutine examines the expression for z_j. It adds variables to Y that are necessary for z_j to have a tight convex envelope in the infinite limit, and it records the bilinear interactions between variables set A.

RequireConvex(integer j)
IF ($j > n$) THEN
  FOR $i := 1$ TO $j - 1$ BEGIN

```
        IF (c_i^j > 0) RequireConvex(i)
        IF (c_i^j < 0) RequireConcave(i)
        IF (¬ Convex(ξ_i^j, z_i, z̄_i))  Y := Y ∪ BaseVars(z_i)
        ELSE RequireConvex(i)
        IF (B_ii^j > 0) RequireConvex(i)
        IF (B_ii^j < 0) Y := Y ∪ BaseVars(z_i)
        FOR k := 1 TO j − 1 BEGIN
            IF (i ≠ k ∧ B_ik^j ≠ 0) THEN
               A := A ∪ (BaseVars(i), BaseVars(k))
               Y := Y ∪ (BaseVars(i) ∩ BaseVars(k))
               IF (B_ik^j > 0) THEN
                  IF (z̄_k > 0) RequireConvex(i)
                  IF (z_k < 0) RequireConcave(i)
                  IF (z̄_i > 0) RequireConvex(k)
                  IF (z_i < 0) RequireConcave(k)
               ELSE
                  IF (z̄_k > 0) RequireConcave(i)
                  IF (z_k < 0) RequireConvex(i)
                  IF (z̄_i > 0) RequireConcave(k)
                  IF (z_i < 0) RequireConvex(k)
               END
            END
        END
    END
END
```

ALGORITHM 4. Subroutine to ensure a tight concave envelope

This is the analogous subroutine for ensuring a tight concave envelope.

```
RequireConcave(integer j)
IF (j > n) THEN
    FOR i := 1 TO j − 1 BEGIN
        IF (c_i^j > 0) RequireConcave(i)
        IF (c_i^j < 0) RequireConvex(i)
        IF (¬ Concave(ξ_i^j, z_i, z̄_i))  Y := Y ∪ BaseVars(z_i)
        ELSE RequireConcave(i)
        IF (B_ii^j > 0) Y := Y ∪ BaseVars(z_i)
        IF (B_ii^j < 0) RequireConcave(i)
        FOR k := 1 TO j − 1 BEGIN
            IF (i ≠ k ∧ B_ik^j ≠ 0) THEN
               A := A ∪ (BaseVars(i), BaseVars(k))
```

```
                    Y := Y ∪ (BaseVars(i) ∩ BaseVars(k))
                    IF (B_ik^j < 0) THEN
                        IF (z̄_k > 0) RequireConvex(i)
                        IF (z_k < 0) RequireConcave(i)
                        IF (z̄_i > 0) RequireConvex(k)
                        IF (z_i < 0) RequireConcave(k)
                    ELSE
                        IF (z̄_k > 0) RequireConcave(i)
                        IF (z_k < 0) RequireConvex(i)
                        IF (z̄_i > 0) RequireConcave(k)
                        IF (z_i < 0) RequireConvex(k)
                    END
                END
            END
        END
    END
```

ALGORITHM 5. Subroutine to choose which side of a bilinear interaction to add to Y

This subroutine decides which variables in a bilinear interaction of the variables in L on the left hand side and variables in R on the right hand side. For the envelope to be tight in the limit, all the variables from one side must be in Y. It chooses the side whose variables not already in Y are linked to most other variables. This choice results in the minimum number of variables being added to Y.

set **ResolveBilinear**(set L, set R)
$L' := L \setminus Y$
$R' := R \setminus Y$
IF ($L' = \emptyset \vee R' = \emptyset$) RETURN $\emptyset$
$L^* := \{x_i : \exists a \in L', \exists (b,c) \in A, (x_i \in b \wedge a \in c) \vee (a \in b \wedge x_i \in c)\} \setminus L'$
$R^* := \{x_i : \exists a \in R', \exists (b,c) \in A, (x_i \in b \wedge a \in c) \vee (a \in b \wedge x_i \in c)\} \setminus R'$
IF ($|L^*| \geq |R^*|$) RETURN L'
ELSE RETURN R'

ALGORITHM 6. Main driver to determine branching variables

When an element of z has a finite upper bound, it requires a tight convex envelope in the relaxation, and likewise when an element of z has a finite lower bound, it requires a tight concave envelope. The main driver makes the necessary additions to Y to provide the necessary tight envelopes for all the constrained elements of Y.

The driver has two main phases. In the first phase, variables are added to Y based on their role in squared terms and ξ while accumulating information about

the bilinear interactions. In the second phase, the bilinear interactions are resolved. The elements of A must be processed in order of increasing size where the size of an element of A is defined as the maximum of the cardinality of its two component sets. When the argmin used in the algorithm below has more than one element, choose any member.

$A := \emptyset$
$Y := \emptyset$
FOR $j := n + 1$ TO N BEGIN
    IF ($z_j^U < \infty$) RequireConvex(j)
    IF ($z_j^L > -\infty$) RequireConcave(j)
END
WHILE $A \neq \emptyset$ BEGIN
    $(L, R) \in$ argmin$_{(a,b)\in A}$ max$\{|a|, |b|\}$
    $Y := Y \cup$ ChooseBilinear(L,R)
    $A := A \setminus \{(L, R)\}$
END

At the end of the main driver, Y contains the set of variables which require branching.

B. Coefficients for example problems

Table IV. Objective function coefficients for example one

i	c_i	i	c_i	i	c_i	i	c_i
1	-1.066890	7	0.5149100	13	-0.2178510	19	-1.5776400
2	1.334450	8	1.6398600	14	1.0153500	20	0.0811166
3	-1.224840	9	0.0678508	15	0.0649806	21	0.0529182
4	-0.534849	10	-0.4161970	16	-1.1314100	22	1.6965300
5	-1.426790	11	0.5154940	17	1.8142800	23	-0.4532200
6	-0.234014	12	-0.0328012	18	-1.8519400		

Table V. Right hand sides for example one

i	b_i	k	d_k	k	d_k
1	1.0498000	1	-1.4089100	11	0.690719
2	-0.9275090	2	1.7586000	12	1.849930
3	-0.0207621	3	1.6076600	13	-0.281364
4	0.5252870	4	0.5655480	14	1.373770
5	-0.1074780	5	1.5045800	15	0.536129
6	1.0312700	6	-1.4236300	16	1.664650
7	0.3278110	7	0.0323101	17	1.084130
8	-1.6362600	8	-1.9086900	18	0.784348
9	1.0860800	9	0.7539460	19	0.618497
10	-1.8557600	10	1.1442900	20	-1.841670

Table VI. Nonzero elements of A for example one

i	j	A_{ij}	i	j	A_{ij}	i	j	A_{ij}
1	1	0.791494	4	22	-0.6831560	8	13	-0.6074750
1	7	1.028900	5	2	1.2722000	8	18	2.3017330
1	12	1.253470	5	9	0.8997950	8	22	1.2365400
1	19	1.196050	5	12	-0.0921251	9	1	1.5216300
2	2	0.316459	5	20	1.3299400	9	9	0.0291254
2	7	-0.767426	6	9	1.2999700	9	11	1.3245300
2	9	-0.319650	6	13	0.1536300	9	18	-1.5178200
3	9	-1.808227	6	15	-1.9864000	10	7	-1.6190100
3	21	0.573109	6	20	-1.8752800	10	10	0.3279450
3	23	1.953350	7	8	-1.1367900	10	11	1.7859800
4	12	-0.746296	7	12	0.4287280	10	22	1.4543600
4	15	-0.650458	7	16	-0.7282910			
4	19	1.764980	7	18	-1.8227200			

Table VII. Nonzero elements of B for example one

k	i	j	B_{kij}	k	i	j	B_{kij}	k	i	j	B_{kij}
1	2	3	0.5998530	6	3	3	-1.245270	14	6	2	0.8437320
1	4	2	1.1618200	6	4	2	1.070480	15	17	2	-1.5720900
1	10	2	0.0968363	6	8	2	1.481690	15	17	3	1.1675800
1	11	1	0.5953090	7	11	1	-0.738157	15	18	1	-1.2703200
1	12	2	0.0157741	7	15	2	-1.998290	15	20	1	-0.1026120
1	16	2	-0.3495940	8	3	1	0.660109	16	7	1	1.4422200
2	3	2	1.5343800	8	13	2	0.297381	16	14	2	1.4503700
2	8	1	0.6215500	8	15	1	0.753913	17	3	2	-1.1957700
2	10	1	1.8293700	9	1	3	-1.704470	17	6	2	1.5247900
2	16	2	-1.1923600	9	6	2	-0.489378	17	7	2	0.7601060
3	5	1	1.6885800	9	15	2	-0.588499	17	18	1	-0.7240830
3	6	2	1.0501500	10	11	2	-1.113930	18	9	3	-1.8605100
3	16	3	0.0280099	11	3	1	1.192800	18	13	3	-1.0674200
3	17	1	-0.1299910	11	10	3	1.908350	19	16	1	-1.9499800
4	4	1	0.4644940	11	13	2	-1.028490	19	20	3	-1.3409600
4	9	3	-1.0991400	11	14	2	1.395810	20	2	1	-1.8119700
4	11	3	1.2386100	12	18	3	-1.569720	20	7	1	-1.1783800
4	19	2	-0.3364220	13	5	2	-0.535376	20	9	2	0.7480300
4	19	3	0.9405890	13	12	1	0.939518	20	12	2	1.4576600
5	10	2	-1.5426700	13	15	2	-1.577420	20	17	3	-0.0330721
5	18	1	1.5738700	14	1	1	-2.771270				

Table VIII. Variable table for example two

Var.	Lower bound	Upper bound	Objective coef.
y_1	4.828313737302301	7.025538314638521	0
y_2	4.422848629194137	6.620073206530357	0
x_1	6.214608098422191	8.411832675758411	0
x_2	6.214608098422191	8.411832675758411	0
x_3	6.214608098422191	8.411832675758411	0
x_4	0	165.5545475362379	$1.191905471317204\times10^{-5}$
x_5	0	65.55454753623789	$1.51697059985826\times10^{-5}$
x_6	0	186.4007582566057	0.003342149157327824
x_7	0	65.55454753623789	0.004253644382053594
x_8	0	213.5992417433942	0.003342149157327824
x_9	0	65.55454753623789	0.004253644382053594
x_{10}	0	234.4454524637621	$1.191905471317204\times10^{-5}$
x_{11}	0	65.55454753623789	$1.51697059985826\times10^{-5}$
x_{12}	0	165.5545475362379	0.003342149157327824
x_{13}	0	86.40075825660574	0.004253644382053594
x_{14}	0	186.4007582566057	0.9371515827914514
x_{15}	0	86.40075825660574	1.192738378098211
x_{16}	0	213.5992417433942	0.9371515827914514
x_{17}	0	86.40075825660574	1.192738378098211
x_{18}	0	234.4454524637621	0.003342149157327824
x_{19}	0	86.40075825660574	0.004253644382053594
x_{20}	0	165.5545475362379	0.00334214915732783
x_{21}	0	113.5992417433942	0.004253644382053602
x_{22}	0	186.4007582566057	0.9371515827914529
x_{23}	0	113.5992417433942	1.192738378098213
x_{24}	0	213.5992417433942	0.9371515827914529
x_{25}	0	113.5992417433942	1.192738378098213
x_{26}	0	234.4454524637621	0.00334214915732783
x_{27}	0	113.5992417433942	0.004253644382053602
x_{28}	0	165.5545475362379	$1.191905471317204\times10^{-5}$
x_{29}	0	134.4454524637621	$1.51697059985826\times10^{-5}$
x_{30}	0	186.4007582566057	0.003342149157327824
x_{31}	0	134.4454524637621	0.004253644382053594
x_{32}	0	213.5992417433942	0.003342149157327824
x_{33}	0	134.4454524637621	0.004253644382053594
x_{34}	0	234.4454524637621	$1.191905471317204\times10^{-5}$
x_{35}	0	134.4454524637621	$1.51697059985826\times10^{-5}$

Table IX. Objective coefficients for example three

Var.	Coef.	Var.	Coef.	Var.	Coef.
y_1	-0.4000	x_6	-0.9196	x_{14}	-0.0108
y_2	-0.5000	x_7	-0.2165	x_{15}	-0.8976
y_3	-0.6000	x_8	-0.0944	x_{16}	-0.8341
x_1	-0.4239	x_9	-0.8582	x_{17}	-0.7077
x_2	-0.9664	x_{10}	-0.0174	x_{18}	-0.6108
x_3	-0.4955	x_{11}	-0.6566	x_{19}	-0.7102
x_4	-0.8139	x_{12}	-0.2996	x_{20}	-0.2678
x_5	-0.8432	x_{13}	-0.2621		

C. Solution for example problems

Table X. Solution of example one

Var.	Value	Var.	Value	Var.	Value
y_1	-0.6032	x_6	0.1085	x_{14}	-1.2352
y_2	-1.9003	x_7	2.0000	x_{15}	-0.0395
y_3	2.0000	x_8	-2.0000	x_{16}	0.0899
x_1	0.9258	x_9	-0.4036	x_{17}	-2.0000
x_2	1.4371	x_{10}	-1.5271	x_{18}	-0.5893
x_3	2.0000	x_{11}	-2.0000	x_{19}	-2.0000
x_4	1.8715	x_{12}	2.0000	x_{20}	-0.2627
x_5	-1.6586	x_{13}	2.0000	Objective	-20.8008

Table XI. Solution of example two

Var.	Value	Var.	Value	Var.	Value
y_1	6.7795	x_{12}	165.56	x_{25}	86.40
y_2	6.0863	x_{13}	86.40	x_{26}	234.46
x_1	7.4726	x_{14}	186.40	x_{27}	73.37
x_2	7.8781	x_{15}	86.40	x_{28}	165.56
x_3	8.1658	x_{16}	213.60	x_{29}	116.43
x_4	165.56	x_{17}	86.40	x_{30}	186.40
x_5	65.55	x_{18}	234.46	x_{31}	103.40
x_6	186.40	x_{19}	73.37	x_{32}	213.60
x_7	65.55	x_{20}	165.56	x_{33}	86.40
x_8	213.60	x_{21}	113.60	x_{34}	234.46
x_9	65.55	x_{22}	186.40	x_{35}	73.37
x_{10}	234.46	x_{23}	103.40	Objective	-183.29
x_{11}	65.55	x_{24}	213.60		

Table XII. Solution of example three

Var.	Value	Var.	Value	Var.	Value
y_1	-2.0000	x_6	2.6414	x_{14}	0.0261
y_2	0.6019	x_7	0.6219	x_{15}	2.1703
y_3	1.8981	x_8	0.2712	x_{16}	2.0167
x_1	1.2176	x_9	2.0750	x_{17}	1.7065
x_2	2.7759	x_{10}	0.0421	x_{18}	1.4728
x_3	1.4233	x_{11}	1.5876	x_{19}	1.7125
x_4	2.3378	x_{12}	0.7244	x_{20}	0.6458
x_5	2.4220	x_{13}	0.6337	Objective	-21.412

References

1. F. A. Al-Khayyal and J. E. Falk. Jointly constrained biconvex programming. *Mathematics of Operations Research*, 8(2):273–286, 1983.
2. I. P. Androulakis, C. D. Maranas, and C. A. Floudas. αBB: A global optimization method for general constrainted nonconvex problems. *Journal of Global Optimization*, 7(4):337–363, 1995.
3. T. G. W. Epperly and R. E. Swaney. Branch and bound for global NLP: New bounding LP. In Ignacio E. Grossmann (ed.), *Global Optimization in Engineering Design*, chapter 1. Kluwer, 1996.
4. J. E. Falk and R. M. Soland. An algorithm for separable nonconvex programming problems. *Management Science*, 15(9):550–569, 1969.
5. R. Horst. An algorithm for nonconvex programming. *Mathematical Programming*, 10:312–321, 1976.
6. R. Horst. Deterministic global optimization with partition sets whose feasibility is not known: Application to concave minimization, reverse convex constraints, DC-programming, and Lipschitzian optimization. *Journal of Optimization Theory and Application*, 58(1):11–37, 1988.
7. R. Horst and N. V. Thoai. Conical algorithm for the global minimization of linearly constrained decomposable concave minimization problems. *Journal of Optimization Theory and Applications*, 74(3):469–486, 1992.
8. R. Horst and N. V. Thoai. Constraint decomposition algorithms in global optimization. *Journal of Global Optimization*, 5:333–348, 1994.
9. M. G. Ierapetritou and E. N. Pistikopoulos. Batch plant design and operations under uncertainty. *Industrial & Engineering Chemistry Research*, 35(2):772–787, 1996.
10. G. P. McCormick. Computability of global solutions to factorable nonconvex programs: Part I — Convex underestimating problems. *Mathematical Programming*, 10:147–175, 1976.
11. G. P. McCormick. *Nonlinear Programming, Theory, Algorithms, and Applications*. John Wiley & Sons, 1983.
12. B. A. Murtagh and M. A. Saunders. MINOS 5.1 user's guide. Technical Report SOL 83-20R, Systems Optimization Laboratory, Stanford University, Stanford, CA 94305-4022, 1987.
13. L. D. Muu. An algorithm for solving convex programs with an additional convex-concave constraint. *Mathematical Programming*, 61:75–87, 1993.
14. L. D. Muu and W. Oettli. Combined branch-and-bound and cutting plane methods for solving a class of nonlinear programming problems. *Journal of Global Optimization*, 3:377–391, 1993.
15. A. Neumaier. *Interval Methods for Systems of Equations*. Encyclopedia of Mathematics and Its Applications. Cambridge University Press, 1990.
16. A. T. Phillips and J. B. Rosen. A parallel algorithm for constrained concave quadratic global minimization. *Mathematical Programming*, 42:421–448, 1988.
17. T. Q. Phong, L. T. H. An, and P. D. Tao. Decomposition branch and bound method for globally solving linearly constrained indefinite quadratic minimization problems. *Operations Research Letters*, 17:215–220, 1995.
18. I. Quesada and I. E. Grossmann. A global optimization algorithm for linear fractional and bilinear programs. *Journal of Global Optimization*, 6:39–76, 1995.
19. H. S. Ryoo and N. V. Sahinidis. Global optimization of nonconvex NLPs and MINLPs with applications in process design. *Computers and Chemical Engineering*, 19(5):551–566, 1995.
20. H. D. Sherali and A. Alameddine. A new reformulation-linearization technique for bilinear programming problems. *Journal of Global Optimization*, 2:379–410, 1992.
21. H. D. Sherali and C. H. Tuncbilek. A global optimization algorithm for polynomial programming problems using a reformulation-linearization technique. *Journal of Global Optimization*, 2:101–112, 1992.
22. H. D. Sherali and C. H. Tuncbilek. A reformulation-convexification approach for solving nonconvex quadratic programming problems. *Journal of Global Optimization*, 7:1–31, 1995.
23. E. M. B. Smith and C. C. Pantelides. Global optimization of general process models. In Ignacio E. Grossmann, editor, *Global Optimization in Engineering Design*, chapter 12. Kluwer, 1996.
24. H. Tuy. Effect of the subdivision strategy on convergence and efficiency of some global optimization algorithms. *Journal of Global Optimization*, 1(1):23–36, 1991.