An Interval Branch and Bound Algorithm for Bound Constrained Optimization Problems

R. Baker Kearfott

Department of Mathematics
University of Southwestern Louisiana

Abstract. In this paper, we propose modifications to a prototypical branch and bound algorithm for nonlinear optimization so that the algorithm efficiently handles constrained problems with constant bound constraints. The modifications involve treating subregions of the boundary identically to interior regions during the branch and bound process, but using reduced gradients for the interval Newton method. The modifications also involve preconditioners for the interval Gauss–Seidel method which are optimal in the sense that their application selectively gives a coordinate bound of minimum width, a coordinate bound whose left endpoint is as large as possible, or a coordinate bound whose right endpoint is as small as possible. We give experimental results on a selection of problems with different properties.

1. Introduction

Interval branch and bound methods have been recognized for some time as a class of *deterministic* methods which will, *with certainty*, find the constrained *global* optima of a function within a box, even when implemented on a machine with finite precision arithmetic. In particular, it is possible with interval arithmetic to

(1) Find, with certainty, all global minima of the nonlinear objective function

$$\phi(X) = \phi(x_1, x_2, \ldots, x_n),$$

where bounds $\underline{x}_i$ and $\overline{x}_i$ are known such that

$$\underline{x}_i \le x_i \le \overline{x}_i \text{ for } 1 \le i \le n.$$

The set of constant bounds in (1) may be succinctly written as the interval vector

$$\mathbf{X} = ([\underline{x}_1, \overline{x}_1], [\underline{x}_2, \overline{x}_2], \ldots, [\underline{x}_n, \overline{x}_n])^T;$$

we will denote the vector of midpoints of these intervals by

$$m(\mathbf{X}) = ((\underline{x}_1 + \overline{x}_1)/2, \ldots, (\underline{x}_n + \overline{x}_n)/2)^T$$

Many interval methods belong to a general class of branch and bound methods; one such method, not using interval arithmetic, is as described in Chapter 6 of [**17**]. Such methods have the following components.

- A technique for partitioning a region $\mathbf{X}$ into smaller regions.
- A technique for computing a lower bound $\underline{\phi}$ of the objective function ϕ over a region $\mathbf{X}$.

In such methods, a region is first partitioned, and $\underline{\phi}$ is computed for the subregion. That subregion (from the list of all subregions which have been produced) corresponding to the smallest $\underline{\phi}$ is then selected for further partitioning. The algorithms terminate when further partitioning does not result in an increase in the underestimate. Such a branch and bound algorithm in the interval context occurs as a method for computing the range of a function in [**14**, p. 49], in [**20**, §3.2], etc.

In such interval branch and bound methods, lower bounds on ϕ are computed in a particularly natural and general way by evaluating ϕ using interval arithmetic; furthermore, such interval function values also lead to upper bounds on ϕ, which may be used to discard some of the subregions, and thus decrease the total number of subregions which must be processed. Also, *interval Newton* methods may be used both to reject interior subregions which do not contain critical points, and to replace subregions by smaller ones via a rapidly converging iteration scheme.

The elements of interval arithmetic underlying such methods are explained well in [**1**], [**14**], [**15**], or [**20**].

Moore, Hansen, Sengupta and Walster, Ratschek and Rokne, and others have spent substantial effort over a number of years in the development of interval methods for global optimization; some of the techniques and results appear in [**4**], [**5**], [**18**], [**19**], and [**20**]. Test results which indicate the competitiveness of such algorithms appear in [**21**] and elsewhere. Treatises on these methods are [**20**] and a forthcoming book of Hansen.

The author and his colleagues have recently developed a technique which, in practice, results in superior behavior of the interval Newton method. The goal of the interval Newton method is to replace the coordinate bounds $\mathbf{x}_i = [\underline{x}_i, \overline{x}_i]$ of a region $\mathbf{X}$ by coordinate bounds $\tilde{\mathbf{x}}_i$ such that the resulting region contains all of the critical points of the original region, but such that the widths of the $\tilde{\mathbf{x}}_i$ are smaller than the corresponding widths of the $\mathbf{x}_i$. The author's *preconditioning* technique gives widths for $\tilde{\mathbf{x}}_i$ which are *optimally small* for a given interval extension[1]; cf. [**10**] and [**16**]. Related preconditioners can give a minimal right endpoint or a maximal left endpoint for $\tilde{\mathbf{x}}_i$; see [**11**] or [**16**]. The author has used this technique in a scheme to reliably find all solutions to nonlinear systems of equations within a box (ibid.).

Interval algorithms for reliably finding all roots to nonlinear systems of equations have a somewhat similar structure to branch and bound optimization algorithms, but differ in some important respects. Nonlinear equation solvers also involve a subdivision process and a search; see [**7**], [**9**], [**15**], and others. However, without an objective function more boxes (containing *all* possible roots) must be considered. Also, nonlinear equation solvers do not need to consider subregions abutting the boundary of the original region specially, since only roots (i.e. critical points), and not optima occurring on boundaries, are of interest.

This paper accomplishes two goals: (i) to indicate how the preconditioning techniques can be included effectively within a global optimization algorithm, and (ii) to develop and test a prototypical structure for an interval algorithm bound-constrained global optimization. For clarity and to study the effects of various components, we have attempted to

[1] By optimality, we mean that the width of a single coordinate in the Gauss–Seidel sweep is minimized over all possible preconditioner rows, given the initial guess point and the interval Jacobi matrix (or slope matrix).

maintain simplicity in the algorithm; production quality algorithms would include more of the techniques in [**5**], [**20**], and in Eldon Hansen's upcoming book.

In §2, we give an overview of the interval Newton method we use, while we catalogue our preconditioners in §3. The modified global optimization algorithm and the variants of the interval Newton algorithm embedded in it appear in §4. Results of numerical experiments are presented in §5, while conclusions appear in §6. Possible future work is outlined in §7.

Throughout, boldface will denote interval scalars, vectors, and matrices. Lower case letters will denote scalar quantities, while vectors and matrices will be denoted with upper case.

2. The Interval Gauss–Seidel method

Interval Newton methods are used in general to sharpen bounds on the solutions to systems of nonlinear equations, and in computational existence and uniqueness tests; see [**14**, ch. 5], [**9**], [**15**], or [**20**], among others. Here, we will use them to efficiently reduce the sizes of interior subregions containing critical points, and to reject subregions which do not contain critical points.

Suppose we have a function $F:\mathbb{R}^n \to \mathbb{R}^n$, i.e.

$$F(X) = (f_1(x_1, x_2, \ldots, x_n), \ldots, f_n(x_1, x_2, \ldots, x_n))^T, \tag{2}$$

suppose $\mathbf{F}(\mathbf{X})$ denotes an inclusion monotonic Lipschitz interval extension[2] of F on the box $\mathbf{X}$, and suppose $\mathbf{F}'(\mathbf{X})$ denotes an inclusion monotonic Lipschitz interval extension of the Jacobi matrix of F on the box $\mathbf{X}$. Then, in an interval Newton method, we first form the linear interval system

$$\mathbf{F}'(\mathbf{X}_k)(\tilde{\mathbf{X}}_k - X_k) \ni -F(X_k), \tag{3}$$

where $X_k = (x_1, x_2, \ldots, x_n)^T \in \mathbf{X}_k$ represents a predictor or initial guess point. There are various methods of formally "solving" (3) using interval arithmetic; in these, the mean value theorem implies that the resulting box $\tilde{\mathbf{X}}_k$ will contain all solutions to $F(X) = 0$ in $\mathbf{X}_k$. Such methods include interval Gaussian elimination, the Krawczyk method, and the interval Gauss–Seidel method; see [**15**] for explanations and references.

The solution method for (3) is better if the widths of the component intervals of $\tilde{\mathbf{X}}_k$ are smaller. From this point of view, the interval Gauss–Seidel method is particularly good; see Theorem 4.3.5 in [**15**]. Furthermore, the interval Gauss–Seidel method generally functions better if we first *precondition* (3), i.e. if we multiply by a non-interval matrix Y to obtain

$$Y\mathbf{F}'(\tilde{\mathbf{X}}_k - X_k) \ni -YF. \tag{3b}$$

Here, we denote the i-th row of the preconditioner matrix Y by Y_i, we set $k_i = Y_i F(X_k)$, and we set

$$\begin{aligned} Y_i\mathbf{F}' = \mathbf{G}_i &= (\mathbf{g}_{i,1}, \mathbf{g}_{i,2}, \cdots, \mathbf{g}_{i,n}) \\ &= ([\underline{g}_{i,1}, \overline{g}_{i,1}], [\underline{g}_{i,2}, \overline{g}_{i,2}], \cdots, [\underline{g}_{i,n}, \overline{g}_{i,n}]). \end{aligned}$$

We then have

[2] See [**14**, ch. 3 and ch. 4] [**15**], or [**20** §**2.6**-§**2.7**].

ALGORITHM 2.1. *(Simplified preconditioned interval Gauss–Seidel) Do the following for $i = 1$ to n.*

1. *(Recompute a coordinate.)*
   (a) *Compute the preconditioner row Y_i.*
   (b) *Compute k_i and $\mathbf{G}_i$.*
   (c) *Compute*

$$\tilde{\mathbf{x}}_i = x_i - \left[k_i + \sum_{\substack{j=1 \\ j \neq i}}^{n} \mathbf{g}_{i,j}(\mathbf{x}_j - x_j) \right] \Big/ \mathbf{g}_{i,i} \tag{4}$$

   using interval arithmetic.
2. *(Update the coordinate.) If $\tilde{\mathbf{x}}_i \cap \mathbf{x}_i = \emptyset$, then signal that there is no root of F in $\mathbf{X}_k$. Otherwise, replace $\mathbf{x}_i$ by $\tilde{\mathbf{x}}_i$.*

The following theorem is part of Theorem 5.18 in [**15**], and had previously been observed in various contexts by various researchers.

THEOREM 2.2. *Suppose $F : \mathbf{X} \subseteq \mathbb{R}^n \to \mathbb{R}^n$ is Lipschitz continuous on $\mathbf{X}$, and suppose $\mathbf{F}'$ is a componentwise interval extension to the Jacobi matrix of F. If $\mathcal{IG}(\mathbf{X})$ is the result of applying Algorithm 2.1 to $\mathbf{X}$, then:*

(i) *Every root $X^* \in \mathbf{X}$ of F satisfies $X^* \in \mathcal{IG}(\mathbf{X})$.*
(ii) *If $\mathcal{IG}(\mathbf{X}) \cap \mathbf{X} = \emptyset$, then F contains no root in $\mathbf{X}$.*
(iii) *If X_k is in the interior of $\mathbf{X}$, $\mathcal{IG}(\mathbf{X}) \neq \emptyset$, and $\mathcal{IG}(\mathbf{X})$ is contained in the interior of $\mathbf{X}$, then F contains a unique root in $\mathbf{X}$.*

This theorem provides a computational existence and uniqueness test which is more practical than the Kantorovich theorem. Also, if $\mathcal{IG}(\mathbf{X})$ is contained in the interior of $\mathbf{X}$, then, typically, iteration of Algorithm 2.1 (reinitializing X_k to the midpoint vector of $\mathbf{X}$, and recomputing $\mathbf{F}'(\mathbf{X})$ each time through) will result in convergence of $\mathbf{X}$ to an approximate point vector which represents sharp bounds on the root. This is an efficient way of obtaining global optima which are interior points in our branch and bound algorithm.

REMARK 2.3. *In practice, we replace $\mathbf{x}_i$ by $\tilde{\mathbf{x}}_i \cap \mathbf{x}_i$ in Step 2 of Algorithm 2.1; it is not hard to show that (i) and (ii) of Theorem 2.2 remain valid when we do so. Property (iii) remains valid under certain conditions; see [**8**] and the clarification thereof in [**16**].*

A preconditioner matrix Y commonly recommended in the literature is the inverse of the matrix of midpoints of the elements of $\mathbf{F}'(\mathbf{X})$; see [**15**, §4.1]. However, we have developed other preconditioners which in many practical situations have advantages. Moreover, different preconditioners in this class can be used to advantage in handling constrained global optimization problems. We give a brief introduction to these preconditioners in the next section.

3. LINEAR PROGRAMMING PRECONDITIONERS

In [**10**], we introduced the concept of *width optimal* preconditioner row Y_i, and presented a technique for computing preconditioners which were either width optimal or which had known small widths. Such computations were based on solving a linear programming problem for the components of each preconditioner row. Numerical results in [**10**] indicated that, despite the cost to obtain the Y_i, these procedures were beneficial to the overall interval Newton method. Subsequent development of low-cost preprocessing ([**6**]), and reformulation of the linear programming problem and its method of solution ([**16**]) led to interval Newton methods which are in many cases several times faster than even that in [**10**].

The width optimal preconditioner rows are part of a class of preconditioner rows which can be computed as solutions of similar linear programming problems; see [**16**] and [**11**]. We define these preconditioners here.

We have classified preconditioners into C-preconditioners and S-preconditioners. Here, we consider only C-preconditioners, both for simplicity and since these have been the most successful in our root-finding experiments. However, S-preconditioners may eventually play a valid rôle in determining that a global optimum occurs on a boundary.

Throughout we will refer to preconditioner rows Y_i as preconditioners, since Y_i may be computed independently (and may indeed be of a different type) for each i.

DEFINITION 3.1. *A preconditioner row Y_i is called a* **C-preconditioner**, *provided* $0 \notin \mathbf{g}_{i,i}$ *in* (4).

Thus, requiring a preconditioner to be a C-preconditioner assures that $\tilde{\mathbf{x}}_i$ in (4) is a single connected interval, and extended interval arithmetic need not be used.

DEFINITION 3.2. *A C-preconditioner $Y_i^{C_W}$ is a W-optimal (width-optimal) C-preconditioner if it minimizes the width $w(\tilde{\overline{x}}_i - \tilde{\underline{x}}_i)$ of the image $\tilde{\mathbf{x}}_i$ in* (4) *over all C-preconditioners.*

DEFINITION 3.3. *A C-preconditioner $Y_i^{C_L}$ is an L-optimal (left-optimal) C-preconditioner if it maximizes the left endpoint $\tilde{\underline{x}}_i$ of the image $\tilde{\mathbf{x}}_i$ in* (4) *over all C-preconditioners.*

DEFINITION 3.4. *A C-preconditioner $Y_i^{C_R}$ is an R-optimal (right-optimal) C-preconditioner if it minimizes right endpoint $\tilde{\overline{x}}_i$ of the image $\tilde{\mathbf{x}}_i$ in* (4) *over all C-preconditioners.*

A situation where a W-optimal preconditioner would be appropriate is illustrated in figure 1. In this figure, we expect the image $\tilde{\mathbf{x}}_i$ to lie within $\mathbf{x}_i$, so that $w(\mathbf{x}_i \cap \tilde{\mathbf{x}}_i)$ is minimum when $w(\tilde{\mathbf{x}}_i)$ is. A situation where an L-optimal preconditioner would be appropriate is illustrated in figure 2. There, we expect the image $\tilde{\mathbf{x}}_i$ to be shifted to the right of $\mathbf{x}_i$, so that $w(\mathbf{x}_i \cap \tilde{\mathbf{x}}_i)$ is minimized when the left endpoint of $\tilde{\mathbf{x}}_i$ is minimized. The R-optimal preconditioner is similar to the L-optimal .

As a simple example of the effects of the three different preconditioners, define $F(X)$: $\mathbb{R}^5 \to \mathbb{R}^5$ by

$$f_i(X) = \begin{cases} x_i + \sum_{j=1}^{n} x_j - (n+1) & \text{for } 1 \le i \le n-1 \\ \prod_{i=1}^{n} x_i - 1 & \text{for } i = n \end{cases}$$

Figure 1. The width-optimal preconditioner $Y_i^{C_W}$ minimizes $w(\tilde{\mathbf{x}}_i)$ over all possible preconditioners Y_i.

Figure 2. The left-optimal preconditioner $Y_i^{C_L}$ maximizes the left endpoint $\underline{\tilde{x}}$ over all preconditioners Y_i.

with initial box and initial guess point:

$$\mathbf{X} = \begin{pmatrix} [0,2] \\ [.5,1.1] \\ [.8,1.2] \\ [.9,1.5] \\ [-2,2] \end{pmatrix}, \quad X = \begin{pmatrix} 1.0 \\ .8 \\ 1.0 \\ 1.2 \\ 0.0 \end{pmatrix}, \mathbf{X} - X = \begin{pmatrix} [-1,1] \\ [-.3,.3] \\ [-.2,.2] \\ [-.3,.3] \\ [-2,2] \end{pmatrix}$$

and thus with function and interval Jacobi matrix

$$F(X) = (-1, -1.2, -1, -.8, -1)^T,$$

$$\mathbf{F}'(\mathbf{X}) \approx \begin{pmatrix} 2 & 1 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 & 1 \\ 1 & 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 1 & 2 \\ [-3.96, 3.96] & [-7.2, 7.2] & [-6.6, 6.6] & [-5.28, 5.28] & [0, 3.96] \end{pmatrix}.$$

(Note: This is Brown's almost linear function.) Suppose we wish to solve for the first coordinate in the Gauss–Seidel step. Then the L-optimal preconditioner is

$$Y_i^{C_L} = (1, 0, 0, -1, 0),$$

and

$$\mathbf{G}_1 = (1, 0, 0, -1, 0), \quad k_1 = -1 - (-.8) = -.2.$$

Applying the preconditioned Gauss–Seidel as in (4) thus gives

$$\tilde{\mathbf{x}}_1 = 1 - \frac{-.2 + (-1)[-.3, .3]}{1} = [.9, 1.5].$$

The R-optimal preconditioner is:

$$Y_i^{C_R} = (1, -1, 0, 0, 0),$$

and

$$\mathbf{G}_1 = (1, -1, 0, 0, 0), \quad k_1 = -1 - (-1.2) = .2,$$

and the preconditioned Gauss–Seidel thus gives

$$\tilde{\mathbf{x}}_1 = 1 - \frac{.2 + (-1)[-.3, .3]}{1} = [.5, 1.1].$$

The W-optimal preconditioner is

$$Y_i^{C_W} = (1, 0, -1, 0, 0),$$

and

$$\mathbf{G}_1 = (1, 0, -1, 0, 0), \quad k_1 = -1 - (-1) = 0.$$

Applying the preconditioned Gauss–Seidel thus gives

$$\tilde{\mathbf{x}}_1 = 1 - \frac{0 + (-1)[-.2, .2]}{1} = [.8, 1.2].$$

The inverse midpoint preconditioner is

$$Y \approx (.8, -.2, -.2, -.2, -.101),$$

and

$$\mathbf{G}_1 \approx ([.6, 1.4], [-.7273, .7273], [-.6667, .6667], [-.5333, .5333].[-.2, .2]),$$

For which preconditioned interval Gauss–Seidel method gives

$$\tilde{\mathbf{x}}_1 \approx [-.3542, 2.684].$$

In the above example, we see that computing both the right optimal and left optimal preconditioners, then intersecting the corresponding images, gives a result which is superior to just applying the width-optimal preconditioner. However, straightforward application of this idea results in twice the amount of computation.

Linear programming problems whose solutions are often the W-optimal, L-optimal, and R-optimal preconditioners appear in [**11**] and in [**16**], while efficient solution techniques for these problems appear in [**16**]. A solution to one of these LP problems is the corresponding *optimal* preconditioner only under certain conditions; however, if these conditions are not met, then it can be shown that the resulting preconditioner is still, in a certain sense, good; see [**11**] and [**16**].

Here, we will not be concerned with the distinction between the solution to the linear programming problems and the corresponding W-optimal, L-optimal and R-optimal preconditioners. We will thus denote the solutions to the linear programming problems by $\overline{Y}_i^{C_W}$, $\overline{Y}_i^{C_L}$, and $\overline{Y}_i^{C_R}$, and assume that they make the width of $\tilde{\mathbf{x}}_i$ small, the left endpoint of $\tilde{\mathbf{x}}_i$ large, and the right endpoint of $\tilde{\mathbf{x}}_i$ small, respectively, and will use these facts in the global optimization algorithm.

To illustrate, a linear programming problem for the width-optimal preconditioner is

$$\text{(5)} \qquad \text{minimize } W(V) = \sum_{j=1}^{n-1} \left(-\sum_{l=1}^{n} v_{l+(n-1)} \underline{f}'_{l,j'} + \sum_{l=1}^{n} v_{l+(2n-1)} \overline{f}'_{l,j'} + v_j \right) w(\mathbf{x}_{j'})$$

subject to

$$v_j \geq \sum_{l=1}^{n} \left(v_{l+(n-1)} - v_{l+(2n-1)}\right) \left(\underline{f}'_{l,j'} + \overline{f}'_{l,j'}\right), \qquad 1 \leq j \leq n-1,$$

$$1 = \sum_{l=1}^{n} v_{l+(n-1)} \underline{f}'_{l,i} - \sum_{l=1}^{n} v_{l+(2n-1)} \overline{f}'_{l,i},$$

and

$$v_j \geq 0 \qquad \text{for } 1 \leq j \leq 3n-1,$$

where $Y_i^{C_W} = (y_1, y_2, \ldots, y_n)$ is defined by

$$y_l = v_{l+(n-1)} - v_{l+(2n-1)}, \qquad 1 \leq l \leq n,$$

where

$$j' = \begin{cases} j & \text{if } \ j < i \\ j+1 & \text{if } \ j \geq i. \end{cases}$$

The left-optimal and right-optimal preconditioners have identical constraints, but a modified objective function; see [**11**] or [**16**].

4. A Variant of the Global Optimization Algorithm

We present the algorithms, which incorporate the W-optimal, L-optimal, and R-optimal preconditioners as well as our scheme for handling the bound constraints, in this section. The algorithm borrows from the prototypical algorithm in [**14 p. 49**].

In addition to its basic structure, our branch and bound algorithm requires

1. the subdivision scheme;
2. bookkeeping to track which of the faces of which sub-boxes lie on the boundary of the original region; and
3. the interval Newton (interval Gauss–Seidel) method, incorporating the preconditioners $\overline{Y}_i^{C_W}$, $\overline{Y}_i^{C_L}$, and $\overline{Y}_i^{C_R}$ in an appropriate manner.

We describe these first.

Definition 4.1. *Our* **bisection scheme** *is a function* $\mathcal{B}(\mathbf{X})$ *which, for an interval vector* $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)^T$, *returns the triplet* $(\mathbf{X}^{(1)}, \mathbf{X}^{(2)}, i)$, *where*

$$\mathbf{X}^{(1)} = (\mathbf{x}_1, \ldots, \mathbf{x}_{i-1}, [\underline{x}_i, (\underline{x}_i + \overline{x}_i)/2], \mathbf{x}_{i+1}, \ldots, \mathbf{x}_n)^T$$

and

$$\mathbf{X}^{(2)} = (\mathbf{x}_1, \ldots, \mathbf{x}_{i-1}, [(\underline{x}_i + \overline{x}_i)/2, \overline{x}_i], \mathbf{x}_{i+1}, \ldots, \mathbf{x}_n)^T.$$

As mentioned in [**20**, p. 75], an appropriate bisection scheme (i.e. one which uses an astute choice of i) can make the branch and bound algorithm more efficient. For the related algorithm which finds all roots of a function within $\mathbf{X}$, Moore suggests four possible $\mathcal{B}$ in [**14 pp. 78–81**]; for the same problem we have found a *maximal smear* scheme to work well in the software [**12**]. In [**4**] as well as throughout [**20**], it is recommended to take the maximal width coordinate, i.e. that i with $(\overline{x}_i - \underline{x}_i)$ maximal, in the optimization algorithm. In the experiments below, we choose i to be the *optimization reduced maximal smear* defined by

$$i = \arg \max_{\substack{1 \leq i \leq n \\ \mathbf{x}_i \not\subseteq \partial \mathbf{X} \\ w(\mathbf{x}_i \geq \epsilon_{\mathbf{X}})}} \left\{ \max\left\{ \left|\underline{\nabla\phi(\mathbf{X})}_i\right|, \left|\overline{\nabla\phi(\mathbf{X})}_i\right| \right\} \right\},$$

for some domain tolerance $\epsilon_{\mathbf{X}}$.

DEFINITION 4.2. *Suppose a box* $\tilde{\mathbf{X}}$ *has been produced by (possibly iterative) application of* $\mathcal{B}$, *starting with the initial box* $\mathbf{X}$ *of* (1). *Then, to each coordinate interval* $\tilde{\mathbf{x}}_j = [\underline{\tilde{x}}_j, \overline{\tilde{x}}_j]$ *of* $\tilde{\mathbf{X}}$ *is associated a* **lower boundary flag** l_j *and an* **upper boundary flag** u_j *such that* $l_j =$ *"true" if and only if* $\underline{\tilde{x}}_j = \underline{x}_j$ *and* $u_j =$ *"true" if and only if* $\overline{\tilde{x}}_j = \overline{x}_j$. *We speak of the* **boundary flag vectors** $L = (l_1, l_2, \ldots, l_n)^T$ *and* $U = (u_1, u_2, \ldots, u_n)^T$.
We also associate a **side flag** s_j *to* $\tilde{\mathbf{x}}_j$, *such that* $s_j =$ *"true" if and only if*
(i) $l_j =$ *"true" or* $u_j =$ *"true"*,
(ii) $\underline{x}_j = \overline{x}_j$,
(iii) *and* $\mathbf{X}_j$ *is a boundary coordinate produced according to the "peeling" process of Definition 4.3 below;*
we speak of the vector $S = (s_1, s_2, \ldots, s_n)$.

For the constrained optimization problem, we must systematically search both the interior of the original box $\mathbf{X}$, as well as its lower-dimensional faces. In fact, when we eg. compute an interval value for ϕ on a lower dimensional face (on which some of the coordinates are not intervals), there is less overestimation. Such facts, as well as a striving to make the algorithm simple, dictate that we treat the lower dimensional faces identically to the n-dimensional boxes in the list $\mathcal{L}$ of boxes produced by the algorithm. We do this with the *peeling* process as follows.

DEFINITION 4.3. *Let a flag vector* $P = (p_1, \ldots, p_n)^T$ *be initially set to* ("false", ..., "false")^T *for the initial box, and let* $\mathbf{X}$ *be the current box to be considered in our algorithm, with current flag vector* P, *current side vector* S, *and current boundary flag vectors* L *and* U. *Let* i *be the smallest coodinate such that* $p_i =$ *"false". Then the* **peel operator** $\mathcal{P}(\mathbf{X})$ *is defined to be*

$$\mathcal{P}(\mathbf{X}) = \begin{cases} \mathbf{X} & \text{if } i \text{ does not exist} \\ \{\mathbf{X}^{(l)}, \mathbf{X}^{(u)}, \mathbf{X}\} & \text{otherwise,} \end{cases}$$

where

$$\mathbf{X}^{(l)} = (\mathbf{X}_1, \ldots, \mathbf{X}_{i-1}, \underline{\mathbf{X}}_i, \mathbf{X}_{i+1}, \ldots, \mathbf{X}_n)^T$$

and

$$\mathbf{X}^{(u)} = (\mathbf{X}_1, \ldots, \mathbf{X}_{i-1}, \overline{\mathbf{X}}_i, \mathbf{X}_{i+1}, \ldots, \mathbf{X}_n)^T.$$

The flag p_i *associated with the image boxes is set to "true", while the flag* s_i *is set to "true" only for* $\mathbf{X}^{(l)}$ *and* $\mathbf{X}^{(u)}$. *The flags* l_i *and* u_i *are set consistently with Definition 4.2.*

Thus, with $\mathcal{P}$, we separate the boundary from boxes which span the entire interior; in the latter we need search only for non-boundary critical points.

DEFINITION 4.4. *If* $\mathbf{X}$ *has corresponding flag vector* $P =$ ("true", ..., "true")^T *and an arbitrary side vector* S, *then the* **reduced system dimension** n_{red} *is the number of entries of* S *which are "false", while the* **reduced function** $\phi_R : \mathbb{R}^{nred} \to \mathbb{R}$ *is formed from* ϕ, *considering those coordinates* $\mathbf{X}_i$ *with* $s_i =$ *"true" to be constant parameters. We similarly define the* **reduced gradient** $\nabla_R \phi$ *and* **reduced Hessian matrix** H_R. *We refer to the system formed from* ϕ_R, $\nabla_R \phi$, *and* H_R *as the* **reduced system**.

The modified interval Gauss–Seidel method may now be easily described.

DEFINITION 4.5. *Let* $\mathbf{X}$ *be a box with corresponding boundary flag vectors* L *and* U, *and assume that* p_i = *"true" for* $1 \le i \le n$. *Then the operator* $\mathcal{IG}_\phi(\mathbf{X}, L, U, S)$ *is defined to be the image of* $\mathbf{X}$ *under Algorithm 2.1 in conjunction with Remark 2.3, applied to the reduced system of Definition 4.4. Also, in Step 1, a) of Algorithm 2.1, we form the preconditioner by*

$$Y_i = \begin{cases} \overline{Y}_i^{C_L} & \text{if} \quad l_i = \text{"true" and } u_i = \text{"false"}, \\ \overline{Y}_i^{C_R} & \text{if} \quad l_i = \text{"false" and } u_i = \text{"true"}, \\ \overline{Y}_i^{C_W} & \text{if} \quad l_i = \text{"false" and } u_i = \text{"false"}, \\ \overline{Y}_i^{C_W} & \text{if} \quad l_i = \text{"true" and } u_i = \text{"true"}. \end{cases}$$

The preconditioners in Definition 4.5 are selected to result in a rapid contraction of the subregion to a critical point if the subregion is interior to the constraint set, and which result in a rapid contraction away from the boundary if the formally interior subregion is on the boundary of the constraint set. This should reduce redundant calculations (on the boundary and in the interior), and aid in rapid rejection of regions not containing critical points.

The list $\mathcal{L}$ of boxes and associated flags produced from $\mathcal{B}$, $\mathcal{IG}_\phi$, and $\mathcal{P}$ is ordered such that the first element on the list is the one most likely to contain the global optimum. The "proper order" for this list is defined in [**14**, p. 49] so that a tuplet

$$(\mathbf{X}^{(1)}, \underline{\phi}(\mathbf{X}^{(1)}), \overline{\phi}(\mathbf{X}^{(1)}), L^{(1)}, U^{(1)})$$

occurs before a tuplet

$$(\mathbf{X}^{(2)}, \underline{\phi}(\mathbf{X}^{(2)}), \overline{\phi}(\mathbf{X}^{(2)}), L^{(2)}, U^{(2)})$$

provided $\underline{\phi}(\mathbf{X}^{(1)}) \le \underline{\phi}(\mathbf{X}^{(2)})$.

As is done in [**5**] and [**19**], we use a point Newton method to attempt to find critical points to high accuracy, to get lower upper bounds on the minimal value of the objective function, in order to eliminate boxes from $\mathcal{L}$ which cannot contain the global minimum. This technique is based on the fact that the classical Newton method (or "globalized" variants such as trust region algorithms) often converges to an approximation from starting points in regions too large for rigorous convergence verification. However, since we are solving a constrained problem, we must make sure that such approximate critical points do not lie outside the original region. We also wish to apply the technique to the reduced systems on the boundary. These considerations, combined with a desire to maintain simplicity, have resulted in

ALGORITHM 4.6 (FOR POINT ESTIMATES).

0. *Input the present box* $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)^T$, *the corresponding boundary indicator variable* S, *a domain tolerance* δ_{point}, *a range tolerance* ϵ_{point}, *and an iteration limit* M_{it}. *(*Note: It is appropriate that δ_{point} be small relative to the minimum box width in the overall branch and bound algorithm.*)*
1. *Form a point vector* $X \in \mathbb{R}^n$ *whose* i*-th component is* $(\underline{x}_i + \overline{x}_i)/2$.

2. Form the reduced point vector $X_{\rm red} \in R^{\rm nred}$ from those entries $\mathbf{x}_i$ of X for which $s_i =$ "false".
3. Compute $\nabla_R \phi(X_{\rm red})$.
4. For $\iota = 1$ to $M_{\rm it}$ **DO**;
   a) Compute $H_R(X_{\rm red})$.
   b) Compute the Newton step $V = (H_R(X_{\rm red}))^{-1} \nabla_R \phi(X_{\rm red})$.
      If this step fails (due to singularity of H_R), then return the midpoint vector of $\mathbf{X}$ as X.
   c) If $\|V\| < \delta_{\rm point}$ then proceed to step 5.
   d) For $\iota \geq 2$, if $\|V\|$ is greater than $\|V\|$ from the previous iteration, then return the midpoint vector of $\mathbf{X}$ as X.
   e) Apply the Newton step: $X_{\rm red} \leftarrow X_{\rm red} - V$.
   f) If any coordinate of $X_{\rm red}$ lies outside the corresponding coordinate bounds of $\mathbf{X}$, then return the midpoint vector of $\mathbf{X}$ as X.
   g) Compute $\nabla_R \phi(X_{\rm red})$.
   h) If $\|\nabla_R \phi(X_{\rm red})\| < \epsilon_{\rm point}$ then proceed to step 5.

   END DO.
5. (Return an approximate minimizer.)
   a) Reset those components x_i of X with $s_i =$ "false" to the corresponding components of the computed $X_{\rm red}$.
   b) Return X and ι.

We may now present our main global optimization algorithm.

ALGORITHM 4.7 (BRANCH AND BOUND).

0. Input
   a) the initial box $\mathbf{X}$,
   b) a minimum box width $\epsilon_{\mathbf{X}}$, and
   c) a gradient tolerance ϵ_{∇}.
1. (Initialization)
   a) $b_u \leftarrow \overline{\phi}(\mathbf{X})$.
   b) (Initialize boundary flags)
      (i) $l_j \leftarrow$ "true", $j = 1, \ldots, n$.
      (ii) $u_j \leftarrow$ "true", $j = 1, \ldots, n$.
      (iii) $s_j \leftarrow$ "false", $j = 1, \ldots, n$.
      (iv) $p_j \leftarrow$ "false", $j = 1, \ldots, n$.

   DO WHILE ($\mathcal{L} \cap \{\mathbf{X} \mid \max_{1 \leq j \leq n} w(\mathbf{x}_j) > \epsilon_{\mathbf{X}}\} \neq \emptyset$).

   2. **IF** $\mathcal{P}(\mathbf{X}) \neq \mathbf{X}$, **THEN**
      a) Insert $\mathbf{X}^{(l)}$, $\mathbf{X}^{(u)}$ and $\mathbf{X}$ from $\mathcal{P}$ in the proper order in $\mathcal{L}$ via Algorithm 4.8.
      b) progress $\leftarrow$ "true".
      c) Jump to step 4.

      END IF
   3. (Interval Newton method)
      a) Compute $\nabla_R \phi(\mathbf{X})$.
         IF $0 \notin \nabla_R \phi(\mathbf{X})$ **THEN**

(i) $\underline{\text{progress}} \leftarrow$ "true".
(ii) Proceed to step 4.

END IF

b) Compute $H_R(\mathbf{X})$.

c) Compute $\nabla_R \phi(X)$, where X is the midpoint vector of $\mathbf{X}$.

d) $\mathbf{X} \leftarrow \mathcal{IG}_\phi(\mathbf{X}, L, U, S)$.

(Note: We do not apply this operation if the preconditioner as in Definition 4.5 cannot be found; thus, this operation will return at most one box.)

e) **IF** $\mathcal{IG}_\phi$ in step 4d) has changed $\mathbf{X}$, **THEN**
$\underline{\text{progress}} \leftarrow$ "true"
ELSE
$\underline{\text{progress}} \leftarrow$ "false"
END IF

f) If $\mathbf{X} \neq \emptyset$ then place $\mathbf{X}$ into its proper position in $\mathcal{L}$ via Algorithm 4.8.

4. Pop the first box $\mathbf{X}$ whose coordinate widths all exceed $\epsilon_{\mathbf{X}}$ from $\mathcal{L}$ and place it in $\mathbf{X}$.
Exit if there are no such boxes.

5. (Possible bisection) **IF** $\underline{\text{progress}} =$ "false" **THEN**

a) $(\mathbf{X}^{(1)}, \mathbf{X}^{(2)}, i) \leftarrow \mathcal{B}(\mathbf{X})$.

b) For $i = 1, 2$: If $0 \in \nabla_R \phi(\mathbf{X}^i)$ then place $\mathbf{X}^i$ into its proper position in $\mathcal{L}$.

c) $\underline{\text{progress}} \leftarrow$ "true".

d) Return to step 4.

ELSE

a) Pop the first box $\mathbf{X}$ whose coordinate widths all exceed $\epsilon_{\mathbf{X}}$ from $\mathcal{L}$ and place it in $\mathbf{X}$.
Exit if there are no such boxes.

b) Return to step 2.

END IF

END DO

Besides inserting a box in the proper place in $\mathcal{L}$, the following algorithm also updates the best computed upper bound b_u for the global minimizer and removes boxes which cannot possibly contain a global minimizer from $\mathcal{L}$.

ALGORITHM 4.8 (LIST INSERTION).

0. *Input the list $\mathcal{L}$, the best computed upper bound b_u, the box $\mathbf{X}$ to be inserted, and the corresponding flag vectors L, U, S, and P as in Definitions 4.2 and 4.3.*
1. *(Update upper bound)*
   a) Apply Algorithm 4.6 to obtain an $X \in \mathbf{X}$.
   b) $b_u \leftarrow \min\left\{b_u, \phi(X), \overline{\phi}(\mathbf{X})\right\}$.
2. *(Locate the point of insertion, if insertion is appropriate)*
   Assume the list is of the form $\left\{\mathbf{X}^{(k)}\right\}_{k=1}^{q}$, where $\underline{\phi}(\mathbf{X}^{(k+1)}) \geq \underline{\phi}(\mathbf{X}^{(k)})$, $1 \leq k \leq q-1$, and where we pretend that $\underline{\phi}(\mathbf{X}^{(q+1)})$ is infinite, and where $\mathbf{X}^{(0)}$ is simply the beginning of the list.

FOR $k = 0$ *to* q **WHILE** $\left(\underline{\phi}(\mathbf{X}^{(k+1)}) < \underline{\phi}(\mathbf{X})\right)$ **DO**:

(Return if box cannot contain a global minimum)

If $\overline{\phi}(\mathbf{X}^{(k)}) < \underline{\phi}(\mathbf{X})$ *then return to Algorithm 4.7.*

END DO

3. *(Actual insertion process)*
   a) *Insert* $\mathbf{X}$ *between* $\mathbf{X}^{(k)}$ *and* $\mathbf{X}^{(k+1)}$ *(so that* $\mathbf{X}$ *becomes* $\mathbf{X}^{(k+1)}$*, etc.)*
   b) *Insert* $\underline{\phi}(\mathbf{X})$, $\overline{\phi}(\mathbf{X})$, L, U, S, *and* P *in corresponding lists.*
   c) *Flag* $\mathbf{X}^{(k+1)}$ *(i.e.* $\mathbf{X}$*) according to whether its scaled coordinate widths are all less than* $\epsilon_{\mathbf{X}}$.
4. *(Cull the list of boxes which cannot contain global minima)*
   FOR $j = k+1$ *to* q*: If* $\underline{\phi}(\mathbf{X}^{(j)}) > \overline{\phi}(\mathbf{X})$ *then delete* $\mathbf{X}^{(j)}$ *from* $\mathcal{L}$.

In practice, $\mathcal{L}$ is a linked list, and the operations in Algorithm 4.8 proceed accordingly.

5. Numerical Experiments

We used Algorithm 4.7 in conjunction with the interval Gauss–Seidel procedure as implemented in [**12**] as a framework for implementation. The resulting Fortran 77 program is available from the author for verification of these experiments, and eventually should be polished to production quality.

We chose the following three types of test problems.

(i) A very simple test problem to debug the code.

(ii) Problems which have been used previously in testing global optimization methods. Here, we attempted to select difficult problems which were also fairly easy to program.

(iii) A simply programmable problem which was previously used to test a bound-constrained global optimization algorithm, but which exhibits a difficult singularity.

(iv) A problem which we designed to highlight the differences between the preconditioners we are testing.

Where bound constraints were given previously in the literature, we used these bound constraints.

When the method is generalized to handle more general constraints, (see Section 7 below), we will experiment with additional problems from the excellent test set in [**3**].

The problems in our experiments are as follows.

1. A simple quadratic, $n = 2$.

$$\phi(X) = x_1^2 + (x_2 - 1)^2/2.$$

   Initial box: $[-.1, .1] \times [.9, 1.1]$.
   The gradient system is linear, so the Hessian matrix is a point matrix; the single optimum at $X = (0, 1)^T$ should thus be obtainable in one application of the preconditioned interval Gauss–Seidel algorithm.
2. Problem 1 from [**21**] (three-hump camel back function), $n = 2$.

$$\phi(X) = x_1^2(12 - 6.3x_1^2) + 6x_2(x_2 - x_1).$$

   Initial box: $([-4, 4], [-4, 4])^T$.

This problem has two global optima, at $X = (-4, -2)^T$ and $X = (4, 2)^T$, at which $\phi(X) = -1444.8$. Note the contrast with the optimum $X = (0, 0)^T$, $\phi(X) = 0$ reported in [**21**]; the latter represents a local optimum of the constrained problem.

3. Problem 8 from [**13**], $n = 3$. (This is similar to problem 6 from [**21**], except for the factor of 10 in the first term.)

$$10\sin^2(\pi x_1) + \sum_{i=1}^{n-1} (x_i - 1)^2 \left[1 + 10\sin^2(\pi x_{i+1})\right] + (x_n - 1)^2.$$

Initial box: $([-1.75, 3.25], \ldots, [-1.75, 3.25])^T$.
This problem has a global minimum of $\phi(X) = 0$ at $X = (1, \ldots, 1)^T$, but a large number of local minima which grows exponentially with n. Levy originally proposed this problem to illustrate the effectiveness of his tunneling method at avoiding the numerous local minima, while Walster, Hansen, and Sengupta used it in [**21**] to show that interval methods were also effective at that.

4. The preceeding problem with $n = 4$.
5. The preceeding problem with $n = 5$.
6. The preceeding problem with $n = 6$.
7. The preceeding problem with $n = 7$.
8. The preceeding problem with $n = 8$.
9. A generalization of Powell's singular function, as given in [**2**], $n = 4$.

$$\phi(X) = \sum_{i \in J} \Big[(x_i + 10x_{i+1})^2 + 5\,(x_{i+2} - x_{i+3})^2 + (x_{i+1} - 2x_{i+2})^4 + 10\,(x_i - 10x_{i+3})^4 \Big],$$

where $J = \{1, 5, 9, \ldots, n-3\}$.
Initial box: $([-1, 1], \ldots, [-1, 1])^T$.
The global optimum is $\phi(X) = 0$, attained at $X = (0, \ldots, 0)^T$; however, the Hessian matrix is of rank 2 at this point. Note that, for $n = 4k$, $k > 1$, the Hessian matrix has k identical diagonal blocks. Various algorithms may or may not take advantage of this, depending on their sophistication.

10. The same as the preceeding problem, but with initial box equal to $([.1, 1.1], \ldots, [.1, 1.1])^T$. This problem has a unique global optimum $\phi(X) \in [2.77, 2.84]$, attained at a unique point $X = (x_1, x_2, x_3, x_4)^T$ with $x_1 \in [.564, .574]$ and $x_2 = x_3 = x_4 = .1$. Note that this contrasts with the computed "constrained" optimum reported in [**2**], in which the same bound constraints were used.
11. A function designed to highlight the difference between the inverse midpoint and the linear programming preconditioners; the gradient is linear in all components except the last one, which is highly nonlinear.

$$\phi(X) = \frac{1}{2}\left\{ \sum_{i=1}^{n} x_i^2 + \left(\sum_{i=1}^{n} x_i \right)^2 \right\} + \sin^2(1000 n x_n).$$

Initial box: $([-1,1],\ldots,[-1,1])^T$; $n=4$.
The unique global minimum within the box is at $X=(0,\ldots,0)^T$.
The Hessian matrix for this function is constant except for the element in the n-th row and n-th column. This Hessian matrix is a symmetric analogue to the Hessian matrix for Brown's almost linear function (cf. the experiments in [**10**]); in the latter, the Jacobi matrix is constant except for the elements in the last row, each of which is highly nonlinear.

In the experiments, $\epsilon_{\mathbf{X}}$ was taken to be 10^{-3} and ϵ_∇ was taken to be 0 in Algorithm 4.7. In the point estimate algorithm, we took $\delta_{\text{point}}=\epsilon_{\text{point}}=10^{-20}$ and $M_{\text{it}}=20$.

In addition to the preconditioner choice strategy embodied in Definition 4.5, we tried three other strategies, which we enumerate here.

Strategy 1: Choose Y_i as in Definition 4.5.

Strategy 2: Choose $Y_i=\overline{Y}_i^{C_W}$ always.

Strategy 3: Choose Y_i to be the i-th row of the inverse of the midpoint matrix of the Hessian matrix. (This has been a common choice throughout the literature.)

Strategy 4: Choose Y_i the opposite of Definition 4.5:

$$Y_i=\begin{cases}\overline{Y}_i^{C_L} & \text{if } l_i=\text{“false” and } u_i=\text{“true”}\\ \overline{Y}_i^{C_R} & \text{if } l_i=\text{“true” and } u_i=\text{“false”}\\ \overline{Y}_i^{C_W} & \text{if } l_i=\text{“false” and } u_i=\text{“false”},\\ \overline{Y}_i^{C_W} & \text{if } l_i=\text{“true” and } u_i=\text{“true”}.\end{cases}$$

We implemented the algorithms in Fortran 77, borrowing from the algorithms and basic interval arithmetic package in [**10**] and [**12**] where possible; we tried all four strategies on all eleven problems on an IBM 3090. For each problem, we kept a record of

- **CPU** the central processing unit time in seconds to complete Algorithm 4.7;
- **DM** the maximum number of boxes in the list $\mathcal{L}$;
- **NFT** the number of interval Newton method calls (including steps 2 and 3 of Algorithm 4.7);
- **NGCR** the number of interval gradient component evaluations to determine the range of the gradient;
- **NJR** the number of evaluations of a row of the interval Jacobi matrix;
- **NGCN** the number of interval gradient component evaluations for the interval Gauss–Seidel method;
- **NPHI** the number of interval objective function evaluations;
- **NBIS** the number of bisections; and
- **CPP** the total CPU time in the point Newton method.

The indicator *CPP* is meant to measure whether a more sophisticated strategy than applying the point Newton method to every box and sub-box produced would benefit the algorithm. Also, for ease of coding, the gradient and Hessian matrix in the point Newton method were obtained from the interval gradient and interval Hessian routines, and thus were fairly expensive. However, in no case did the ratio *CPP*/*CPU* exceed about 1/3.

The CPU time should be taken here as a relative measure only, as we were using portable, software-implemented interval arithmetic which did not utilize modern techniques such as in-lining. Performance with machine-optimized interval arithmetic could be an order of magnitude faster.

The results for all eleven problems and Strategy 1 are given in Table 1, while summary statistics (representing the sum of the above measures over all eleven problems) are given for all four strategies in Table 2. Complete statistics for problems 9, 10, and 11 are given in Table 3.

Table 1. Strategy 1.

#	CPU	DM	NFT	NGCR	NJR	NGCN	NPHI	NBIS	CPP
1	0.0	1	3	8	2	6	7	0	0.0
2	0.0	4	10	38	10	22	22	3	0.0
3	10.9	9	61	507	174	447	108	32	4.1
4	21.6	14	82	908	312	924	146	44	7.8
5	33.8	15	90	1283	425	1570	171	56	11.9
6	53.0	20	109	1877	618	2574	208	69	17.7
7	80.2	21	138	2697	917	4165	254	81	25.1
8	104.9	33	143	3325	1080	5584	277	94	32.0
9	35.6	39	2321	21458	9234	27164	3025	497	4.1
10	0.8	5	62	418	146	521	100	27	0.2
11	34.3	16	432	5954	1712	2252	1086	419	9.5
Tot.	375.1	177	3451	38473	14630	45229	5404	1322	112.4

Table 2. Summary for all four strategies.

Str.	CPU	DM	NFT	NGCR	NJR	NGCN	NPHI	NBIS	CPP
1	375.1	177	3451	38473	14630	45229	5404	1322	112.4
2	374.0	177	2675	40290	15526	47850	5638	1326	111.9
3	291.6	213	2245	32007	9392	43784	5069	1954	100.9
4	378.8	177	3735	40771	15766	48506	5699	1327	112.6

Table 3. Strategy comparison on significant problems.

#	CPU	DM	NFT	NGCR	NJR	NGCN	NPHI	NBIS	CPP	Str.
9	35.6	39	2321	21458	9234	27164	3025	497	4.1	1
	38.1	39	2545	23278	10130	29788	3259	501	4.4	2
	17.5	73	1175	15123	4502	22012	2457	1017	3.3	3
	40.1	39	2621	23895	10434	30700	3337	503	4.5	4
10	0.8	5	62	418	146	521	100	27	0.2	1
	0.8	5	62	415	146	518	100	27	0.2	2
	0.6	5	52	382	103	397	100	36	0.2	3
	0.5	5	46	279	82	262	83	26	0.2	4
11	34.3	16	432	5954	1712	2252	1086	419	9.5	1
	33.7	16	432	5954	1712	2252	1086	419	9.4	2
	35.3	18	531	7636	2100	10500	1471	525	10.5	3
	34.4	16	432	5954	1712	2252	1086	419	9.5	4

In all cases, Algorithm 4.7 completed successfully.

The above results lead to the following conclusions.

The choice of preconditioner is not as important in global optimization problems as in general nonlinear systems.

Overall, the experiments seem to refute our thesis that an appropriate preconditioner will improve the performance of a global optimization algorithm. This is in sharp contrast to experiments reported in [**10**] and [**11**], in which optimal preconditioners never appreciably reduced performance (from the point of view of CPU time and other measures), and in many cases increased performance by orders of magnitude. This is probably partially due to the symmetry in the interval Hessian matrix. Our optimal preconditioners will avoid working with a row in which there is a large amount of overestimation in the intervals (due to eg. a highly nonlinear function component). However, such overestimation in a row of an interval Hessian matrix must also occur in the corresponding column; thus, the overestimation will be present in the sum (4) in Algorithm 2.1 regardless of whether or not that row is included in the linear combination forming $\mathbf{G}_i$. We do note, however, that, in the problem we designed to highlight our preconditioners, our preconditioners resulted in significantly less interval Hessian row evaluations than the inverse midpoint preconditioner.

A second explanation for the lack of performance improvement is that the interval Newton method itself is less important in global optimization algorithms than in general nonlinear system solvers. In particular, use of the upper bound b_u on the global optimum to eliminate boxes seems to be powerful, and it is enhanced in our context by our lower-dimensional searches (with correspondingly smaller overestimations in the interval arithmetic) on boundaries. The relative unimportance of the interval Newton method is suggested by the fact that the total number of bisections and the maximum list size *was* larger when the inverse midpoint preconditioner was used. This seems to indicate that, in that case, the boxes needed to be smaller in order for the interval Gauss–Seidel method to further reduce the coordinate intervals. (Also, see the italicized comment below.)

Use of the point Newton method (Algorithm 4.6) to improve b_u is very worthwhile.

With a very inefficient implementation of the point Newton method, and applying the point Newton method at every conceivable point where it could benefit, it took no more than about 1/3 of the total CPU time. Furthermore, in preliminary experiments in which we did not use the point Newton method (not reported here), the algorithm could not complete Problem 9 without storing at least 4000 boxes in $\mathcal{L}$; when using the inverse midpoint preconditioner, the algorithm also failed on Problem 11 for the same reason. (*It is interesting to note, however, that even without the point Newton method, Strategy 1 completed successfully in about 25 CPU seconds, whereas use of the inverse midpoint preconditioner failed in about 708 CPU seconds.*)

Finally even when the algorithm without the point Newton method completed, the final boxes in $\mathcal{L}$ were of lower quality in the sense that there were clusters of numerous boxes about global minima, some of which were fairly far from the actual global minimum. Finally, CPU times were much larger.

Use of reduced gradients is worthwhile.

We do not have solid statistics on this aspect. However, the bound constraints fit very naturally (without appreciable extra overhead and without complicating the code) into both the interval Newton method and the point Newton method for obtaining point estimates. Furthermore, as mentioned, zero-width coordinates lead to less overestimation in the inverval values. Thus, the presence of bound constraints makes the problem easier than the global unconstrained problem.

7. Future Improvements

It is also be possible to apply this method to several classes of more general constraints than those in (1). For example, differentiable nonlinear constraint functions can be handled in two ways as follows. Suppose we have an inequality constraint of the form

$$h(X) \leq 0.$$

Then we may compute interval extensions $\mathbf{h}(\mathbf{X})$ of h over boxes $\mathbf{X}$ between steps 2 and 3 of the list insertion algorithm (Algorithm 4.8), and return without inserting a box $\mathbf{X}$ if $\underline{h}(\mathbf{X}) > 0$.

We may also include a linear interval equation of the form

$$\nabla \mathbf{h}(\mathbf{X})(\mathbf{X} - X) = \mathbf{h}(\mathbf{X}) \cap (-\infty, 0]$$

in the reduced gradient system, as suggested by Novoa ([**16**]). This transforms the linear interval system into a system with more equations than unknowns, but our linear programming preconditioner technique applies equally well to rectangular systems. In particular, the LP problem (5) may be modified by increasing the index bound on l, and another simple modification allows inclusion of interval right-hand-sides. Any feasible critical points must then necessarily be contained in the box obtained from the resulting preconditioned Gauss–Seidel sweep.

Finally, standard techniques such as use of Lagrange multipliers or the Kuhn-Tucker conditions may be used.

References

1. Alefeld, Götz, and Herzberger, Jürgen, "Introduction to Interval Computations," Academic Press, New York, etc., 1983.
2. Conn, Andrew R., Gould, Nicholas I. M., and Toint, Philippe L., *Testing a Class of Methods for Solving Minimization Problems with Simple Bounds on the Variables*, Math. Comp. **50** 182 (1988), 399–430.
3. Floudas, C. A. and Pardalos, P. M., "A Collection of Test Problems for Constrained Global Optimization Algorithms," Springer-Verlag, New York, 1990.
4. Hansen, E., *Global Optimization Using Interval Analysis-the Multi-Dimensional Case*, Numer. Math. **34** 3 (1980), 247–270.
5. Hansen, E., *An Overview of Global Optimization Using Interval Analysis*, in "Reliability in Computing," Academic Press, New York, 1988, pp. 289–308.
6. Hu, C.-Y., *Preconditioners for Interval Newton Methods*, Ph.D. dissertation, University of Southwestern Louisiana 1990.
7. Kearfott, R. B., *Abstract Generalized Bisection and a Cost Bound*, Math. Comp. **49** 179 (1987), 187–202.
8. Kearfott, R. B., *Interval Newton / Generalized Bisection When There are Singularities near Roots*, Annals of Operations Research **25** (1990), 181–196.

9. Kearfott, R. B., *Interval Arithmetic Techniques in the Computational Solution of Nonlinear Systems of Equations: Introduction, Examples, and Comparisons*, in "Computational Solution of Nonlinear Systems of Equations," (Lectures in Applied Mathematics, volume 26), American Mathematical Society, Providence, RI, 1990, pp. 337–358.
10. Kearfott, R. B., *Preconditioners for the Interval Gauss-Seidel Method*, SIAM J. Numer. Anal. **27** 3 (1990), 804–822.
11. Kearfott, R. B., Hu, C. Y., Novoa, M. III, *A Review of Preconditioners for the Interval Gauss–Seidel Method*, preprint, 1990, Interval Computations.
12. Kearfott, R. B., and Novoa, M., *INTBIS, A Portable Interval Newton/Bisection Package*, ACM. Trans. Math. Software **16** 2 (1990), 152–157.
13. Levy, A. V. and Gomez, S., *The Tunneling Method Applied to Global Optimization*, in "Numerical Optimization 1984," SIAM, Philadelphia, 1984, pp. 213–44.
14. Moore, Ramon E., "Methods and Applications of Interval Analysis," SIAM, Philadelphia, 1979.
15. Neumaier, A., "Interval Methods for Systems of Equations," Cambridge University Press, Cambridge, England, 1990.
16. Novoa, M., *Linear Programming Preconditioners for the Interval Gauss–Seidel Method and their Implementation in Generalized Bisection*, Ph.D. dissertation, University of Southwestern Louisiana.
17. Pardalos, P. M., and Rosen, J. B., "Constrained Global Optimization: Algorithms and Applications," Springer-Verlag, New York, 1987.
18. Ratschek, H., *Inclusion Functions and Global Optimization*, Math. Programming **33** 3 (1985).
19. Ratschek, H., and Rokne, J. G., *Efficiency of a Global Optimization Algorithm*, SIAM J. Numer. Anal. **24** 5 (1987), 1191–1201.
20. Ratschek, H., and Rokne, J., "New Computer Methods for Global Optimization," Wiley, New York, 1988.
21. Walster, G. W., Hansen, E. R. and Sengupta, S., *Test Results for a Global Optimization Algorithm*, in "Numerical Optimization 1984," SIAM, 1985, pp. 272–287.

Keywords. nonlinear algebraic systems, Newton's method, interval arithmetic, Gauss-Seidel method, global optimization, singularities
1980 *Mathematics subject classifications*: Primary: 65K10; Secondary: 65G10

U.S.L Box 4-1010, Lafayette, LA 70504