GLOPT – A Program for Constrained Global Optimization

S. Dallwig, A. Neumaier and H. Schichl
Institut für Mathematik, Universität Wien
Strudlhofgasse 4, A-1090 Wien, Austria
WWW: http://solon.cma.univie.ac.at/
email: {sdal,neum}@cma.univie.ac.at, herman@esi.ac.at

February 1996, revised April 1996

Abstract. GLOPT is a Fortran77 program for global minimization of a block-separable objective function subject to bound constraints and block-separable constraints. It finds a nearly globally optimal point that is near a true local minimizer. Unless there are several local minimizers that are nearly global, we thus find a good approximation to the global minimizer.

GLOPT uses a branch and bound technique to split the problem recursively into subproblems that are either eliminated or reduced in their size. This is done by an extensive use of the block separable structure of the optimization problem.

In this paper we discuss a new reduction technique for boxes and new ways for generating feasible points of constrained nonlinear programs. These are implemented as the first stage of our GLOPT project. The current implementation of GLOPT uses neither derivatives nor simultaneous information about several constraints. Numerical results are already encouraging. Work on an extension using curvature information and quadratic programming techniques is in progress.

Key words: global optimization, global minimum, branch and bound, block separable, feasibility, tunneling, reduce

1991 MSC codes: primary 90C26; secondary 65K05

1. Generalities

Optimization is one of the basic mathematical techniques that lie at the heart of modern competitive management, design and development. The oldest and best developed part, linear programming, is an indispensable planning tool once the interrelations of tasks is complex enough. Aircraft scheduling, the structural design of buildings or bridges, the design of chemical production plants, the operation of electric power plants, the analysis of electrical circuits, and many other problems can, however, be modeled only through nonlinear relationships.

Except in special circumstances, this introduces the potentiality of multiple minima, and in many cases, such multiple minima are indeed known to occur. The optimum is in this case the *global* minimum. The *Global Optimization Home Page* on the World Wide Web (WWW) with URL `http://solon.cma.univie.ac.at/~neum/glopt.html` currently has over 400 commented links to information relevant to global optimization. (`http://cad.bu.edu/go` is the address of another useful global optimization page.)

Methods for finding a local minimum were traditionally globalized simply by repeatedly optimizing from various starting points, hoping that one of them is close enough to the global minimizer so that local optimization reaches it. Using suitable statistical techniques, this approach can be refined further (DIXON & SZEGÖ [8], MOCKUS [29], TÖRN & ŽILINSKAS [41]). More recently, a number of other techniques like simulated annealing (KIRKPATRICK et al. [24]) and genetic algorithms (HOLLAND [13], DAVIS [6]) have been developed, using analogies to physics and biology to approach the global optimum. All these techniques have in common the fact that they tend to find better and better local minima, but – especially when there are lots of minima – not necessarily the global one. Moreover, they become slower and slower as one tries to increase the probability of success.

In combinatorial optimization, where also global optima are wanted but the variables are discrete and take a few values only, these techniques have the same potential and draw-backs. However, for combinatorial problems there is a methodology, called *branch and bound* (see, e.g., NEMHAUSER & WOLSEY [31]) that permits a systematic exploration of the full configuration space of allowed combinations of variables, and yet avoids in many cases the exponential amount of work that a naive exhaustive search would require. The basic idea is that the configuration space is split recursively into smaller and smaller parts (*branch*ing). Tests based on lower *bounds* on the objective eliminate large portions of the configuration space early in the computation so that only a tiny part of the resulting tree of subproblems must be generated and processed.

The same idea can be made to work also in the continuous case, where the variables may take a continuous range of values (PARDALOS & ROSEN [36], HORST & PARDALOS [14]). Using techniques of interval analysis (MOORE [30], NEUMAIER [33]), it is possible to find lower bounds for the objective over extended boxes in configuration space, and the branching technique can be adapted from the combinatorial situation by splitting such boxes. Indeed, interval branch and bound methods have been successfully used on the related problem of finding all solutions of nonlinear systems, and also in unconstrained global optimization (HANSEN [12]).

However, many practical problems involve general linear or nonlinear constraints, and until recently there were no programs that can solve such problems except in very small dimensions or special situations (like indefinite quadratic programs or concave programs). Only recently, general purpose constrained global optimization programs that do not restrict the form of the constraints are becoming available (Epperly [9], Kearfott [23], Maranas & Floudas [27, 28], Schnepper & Stadtherr [39]). GLOPT is a long-term project aiming at the development of software for fast and reliable constrained optimization by exploiting the special structure of the problems so that higher-dimensional problems become tractable.

The present paper concentrates on the zero-order techniques used in GLOPT. For maximal efficiency, second-order techniques are needed, but these will be discussed elsewhere.

2. The GLOPT global optimization program: Overview

GLOPT is a branch and bound algorithm designed to solve the following optimization problem: Find the global minimizer of a block-separable objective function subject to bound constraints and block-separable constraints of the form

$$\sum_k f_k(x_{J_k}) \in [q] \tag{1}$$

or

$$\sum_k f_k(x_{J_k}) + \beta = x_j, \tag{2}$$

where x_{J_k} is a subvector indexed by a one- or two-dimensional index list J_k of size n_k, $f_k : \mathbb{R}^{n_k} \to \mathbb{R}$, $\beta \in \mathbb{R}$, and $[q]$ is a possibly unbounded interval. The objective function has the form of the left-hand side of (1).

Internally, slack variables are introduced to bring the problem into our standard form

$$\begin{array}{ll} \min & x_i \\ \text{s.t.} & F(x) = 0,\ x \in [x]^{\text{init}} \end{array} \tag{3}$$

where a *box* $[x]$ is defined as

$$[x] := [\underline{x}, \overline{x}] = \{\tilde{x} \in \mathbb{R}^n | \underline{x} \le \tilde{x} \le \overline{x}\}, \tag{4}$$

and inequalities are understood componentwise. The internal structure of the vector-valued function F resulting from transforming (1) and (2) to the form (3) is retained.

For processing with GLOPT, constrained optimization problems are coded in the input format NOP (Neumaier[35]) that explicitly displays

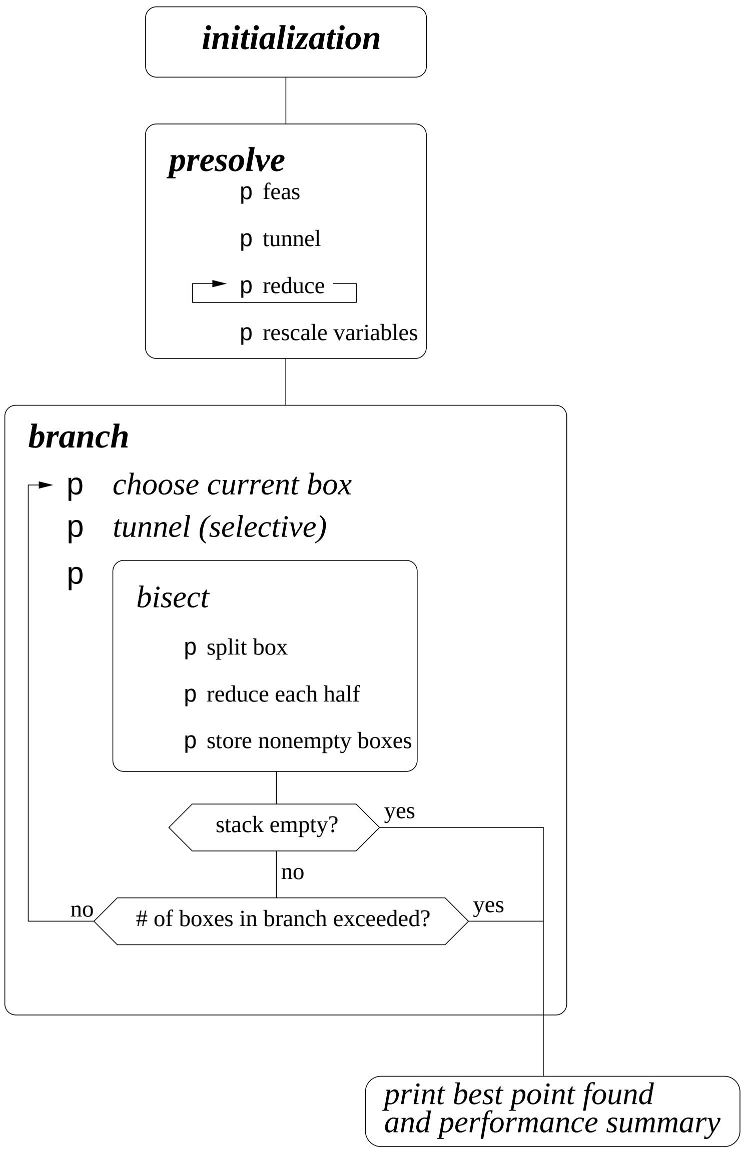

Figure 1. Flowchart of GLOPT

the internal structure of the problem with very little overhead. The NOP reader (see Section 3) accepts the more general formulation of the problem in terms of (1) and (2) and transforms it into an internal representation in standard form.

GLOPT is written in Fortran77; since it does not use directed rounding, its reliability is that expected of other numerically stable floating point calculations; i.e., because of rounding errors, we find a nearly globally optimal point that is near a true local minimizer. Unless there are several local minimizers that are nearly global, we thus find a good approximation to the global minimizer. (With proper directed rounding, we could even achieve mathematical guarantees, at the expense of machine dependence of the program and a time penalty for frequent switching of rounding modes.)

GLOPT splits the problem recursively into subproblems that are either eliminated or reduced in their size, making extensive use of the block separable structure of the optimization problem. As in a method by Jansson & Knüppel [19] for bound constrained global optimization, no derivative information is used in GLOPT.

This information is used in three separate tasks called **feas**, **tunnel** and **reduce**. They are novel improvements of techniques known from the literature, and are superior to the simple bound-and-discard techniques of traditional (combinatorial) branch and bound algorithms.

Branching is done by splitting a box along the widest coordinate into two smaller boxes; these boxes are kept on a stack until they are too small to be processed further. The basic step of GLOPT is the work done on each box; the order in which the boxes are processed is determined by heuristics ensuring a proper balance between a depth first and a breadth first search in the tree of boxes defined by the branching process. We tested a number of options available and believe to have found an efficient way of achieving this.

A rough outline of the structure of GLOPT is given in the flow chart of Figure 1. The current implementation is further enhanced by a number of smaller features improving performance, such as details of box selection and branching strategy. Possible techniques for doing this are discussed, e.g., in [3, 5, 18, 23, 37, 38], mainly for the bound constraint case. More general constraints may need deviations from these principles. Since our current implementaion is preliminary and it is unclear to which extend these will have to be modified to work together with the sequential quadratic programming and curvature exploiting techniques currently under investigation, we refrain from giving details here but concentrate on the *new* zero-order features of GLOPT.

3. NOP: Problem specification

Our global optimization algorithm requires that full details of the problem structure are available to those parts of the algorithm that provide global information by using analytic estimates. Therefore, we had to decide on a uniform way to provide this information. For this purpose, we designed and implemented a specification language for the input of global optimization problems called NOP. The documentation of the current version NOP1.0 of the specification format (NEUMAIER [35]) is available under the URL

`http://solon.cma.univie.ac.at/~neum/glopt/nop.html`

on our WWW server.

The main feature of NOP is that it splits the problem description into a number of small uniform units called elements; currently there are about 30 different types of elements that allow an efficient coding of the major types of formulas occuring in typical optimization problems. Moreover, the user can specify new elements that are not predefined.

As a check for its usefulness and user friendliness, and simultaneously as a basis for testing various versions of our algorithm, we coded in NOP over 60 test problems from three well-known collections [10, 19, 44] of global optimization test problems. While NOP is already in a very useful stage, it still has some inefficiencies, and we are experimenting with some modifications.

We developed a NOP reader that translates the user-friendly external NOP format into an internal code that is used by GLOPT to calculate function values, ranges, and other estimates needed. Currently, the NOP reader handles most elements correctly, but there are still a few bugs that lead to the rejection of some of the coded test problems.

GLOPT has facilities to compute function values, ranges, and residuals from the internal problem representation generated by the NOP reader. For each of the many predefined NOP elements (described in [35]) we derived and coded optimal analytical formulas; however, for user-defined elements, ranges are computed by interval arithmetic [33], giving suboptimal bounds.

Thus the only input needed for GLOPT is a NOP file with the problem specification.

4. The search for feasible points

The routines **feas** and **tunnel** are a cheap and a more expensive way of searching for better feasible points. Here we describe the techniques in general terms; for examples see Section 6.

feas works by modifying the absolutely smallest point of the box, sequentially going through the constraints (2) and replacing each previously unused variable x_k on the right hand side by the left-hand side, while checking an approximate satisfaction condition in case the right hand side variable was already used. At the end, the point obtained is checked for lying in the original box. The total work is essentially that of one evaluation of all constraints.

Of course there is no guarantee that **feas** is successful. However, many problems have, in their original formulation, a feasible set with strictly interior points. In such cases, the translation to the block separable format with equality constraints and bound constraints only is achieved by introducing new variables for certain intermediate quantities and slacks. It is easy to see that if the constraints defining these additional variables appear in the ordering in which they would be computed naturally, the resulting block triangular structure implies that **feas** works sucessfully in all cases where the midpoint of the box, restricted to the original set of variables, is feasible in the original formulation. Thus **feas** generalizes the well-known *midpoint test* of interval algorithms for global optimization (e.g., HANSEN [12]).

If **feas** does not work, and in particular always when there are intrinsic equality constraints, the more expensive *tunneling strategy* is tried. The idea is taken from the tunneling method of LEVY & MONTALVO [25] for unconstrained optimization, although the technique used there is heuristic only, and completely different.

tunnel attempts to find a point better than the currently best point by a procedure that can be visualized as tunneling through a mountain to a deeper valley.

Mathematically, one looks for a feasible point of (3) in the subbox $[x]$ by solving the least squares problem

$$\begin{array}{ll} \min & \|F(x)\|_2^2 \\ \text{s.t.} & x \in [x]. \end{array} \tag{5}$$

Since the best point was already used to adapt the bounds of the component containing the objective function value, any solution with $F(x) = 0$ in the box is indeed at least as good as the currently best point. Of course, **tunnel** may fail too, by getting stuck in a local minimum or a saddle point, or by reaching a prespecified number of function calls (in our current implementation, $10n^2$ function calls in **presolve**, and $2n^2$ function calls later) without having converged.

For example, suppose that in minimizing

$$x_2 = (x_1 - 1)^2(x_1 - 4)(x_1 - 6) + 40,$$

we have as best point the local minimizer at $x = \binom{1}{40}$ and look at the current subbox $[x] = \binom{[2.5,6.5]}{[10,90]}$, cf. Figure 2. The best function value

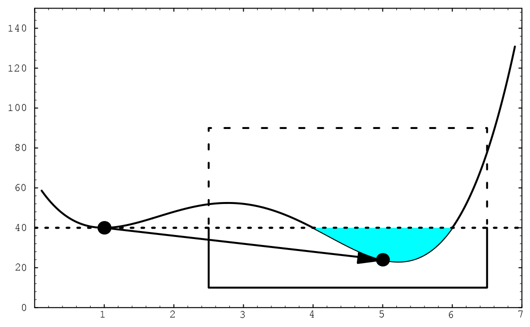

Figure 2. Finding better points by **tunnel**

allows us to reduce $[x_2]$ to $[10, 40]$, and (5) becomes

$$\begin{aligned}&\min\ ((x_1-1)^2(x_1-4)(x_1-6)+40-x_2)^2\\&\text{s.t.}\ \ x_1\in[2.5,6.5],\quad x_2\in[10,40].\end{aligned}\tag{6}$$

A local optimization, if successful, might find any point on the portion of the curve below the dotted line, corresponding to the minimum value 0. For example, the point $x' = \binom{5}{24}$ solves (6).

A slight modification of the least squares problem yields nearly feasible points biased towards the minimum. The following result treats a slightly more general situation with an arbitrary subset C of $\mathbb{R}^n$ in place of a box, and a scaling vector $u > 0$ (of order $O(1)$) whose components give a natural scale to constraint violations.

Theorem. *Let $\hat{f}$ be the global minimum value of*

$$\begin{aligned}&\min f(x)\\&\text{s.t.}\ \ x\in C,\ \ F(x)=0.\end{aligned}\tag{7}$$

If $\underline{f} < \bar{f}$ and $\hat{f} \in [\underline{f}, \bar{f}]$ then any global minimizer $\hat{x}$ of the tunneling problem

$$\begin{aligned}&\min \phi(x) = \varepsilon^2\left(\frac{f(x)-\underline{f}}{\bar{f}-\underline{f}}\right)^2 + \sum_i \left(\frac{F_i(x)}{u_i}\right)^2\\&\text{s.t.}\ \ x\in C\end{aligned}\tag{8}$$

with $\varepsilon > 0$ satisfies

$$f(\hat{x}) \le \hat{f},\quad |F(\hat{x})| \le \varepsilon u.\tag{9}$$

(Absolute values are taken componentwise.)

Proof. Let x^l be a sequence of feasible points of (7) with $f(x^l) \to \hat{f}$ as $l \to \infty$. Then

$$\phi(\hat{x}) \le \lim_{l\to\infty} \phi(x^l) = \varepsilon^2 \left(\frac{\hat{f} - \underline{f}}{\bar{f} - \underline{f}} \right)^2 .$$

Therefore

$$\frac{f(\hat{x}) - \underline{f}}{\bar{f} - \underline{f}} \le \frac{1}{\varepsilon} \sqrt{\phi(\hat{x})} \le \frac{\hat{f} - \underline{f}}{\bar{f} - \underline{f}},$$

so that $f(\hat{x}) \le \hat{f}$. Moreover,

$$\frac{|F_i(\hat{x})|}{u_i} \le \sqrt{\phi(\hat{x})} \le \varepsilon \frac{\hat{f} - \underline{f}}{\bar{f} - \underline{f}} \le \varepsilon,$$

so that $|F(\hat{x})| \le \varepsilon u$. $\square$

In our applications, $f(x) = x_i$ and $C = [x]$ is the current box.

Unfortunately, to get high accuracy one needs a small ε, and that makes the Hessian ill-conditioned. So there is no guarantee that we get close to $\hat{f}$ without resolving this ill-condition by using proper linear algebra techniques. However, unless the $F_i(x)/u_i$ are also badly scaled, just using a standard descent method, we get at least a nearly feasible point, since the ill-condition does not affect the reduction of the sum in (8). Indeed, the problem (5) is the limiting case $\varepsilon \to 0$ of (8).

In our GLOPT implementation, we solve the bound constrained tunneling least squares problem using the PORT routine `nf2b`, an implementation by David Gay based on the NL2SOL code of Dennis et al. [7].

5. Box reduction

The routine **reduce** attempts to shrink a box by elimination of a part of it that can be shown not to contain a point better than the best point found already.

At the moment, this is done equation by equation only, using reduction formulas that are optimal for constraints coded with predefined NOP element functions, and valid but often suboptimal for constraints coded with user-defined element functions. Here the (block) separability is crucial.

In the following, we show how separable constraints can be used to tighten the box constraints. (The block separable case is very similar and will not be discussed explicitly.) The technique is a nonlinear

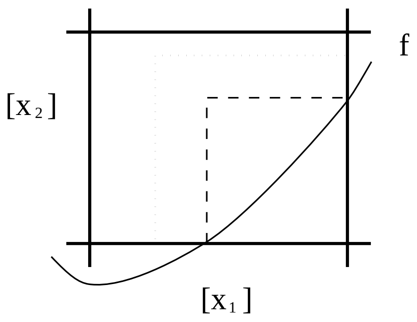

Figure 3. Shrinking a box by **reduce**

version of the generalized Gauss-Seidel method for linear interval equations, introduced in NEUMAIER [32], p.180, and later used by KEARFOTT and his students [21, 22, 40].

For the mathematical formulation we need the *inner addition*

$$[\underline{a},\overline{a}] \overset{\circ}{+} [\underline{b},\overline{b}] := [\underline{a}+\overline{b},\overline{a}+\underline{b}]$$

of intervals. This differs from the standard interval operations, where

$$[\underline{a},\overline{a}] + [\underline{b},\overline{b}] := [\underline{a}+\underline{b},\overline{a}+\overline{b}],$$

$$[\underline{a},\overline{a}] - [\underline{b},\overline{b}] := [\underline{a}-\overline{b},\overline{a}-\underline{b}].$$

Note that $[a]-[b]=[c] \iff [a]=[c] \overset{\circ}{+} [b]$.

Proposition *Suppose that*

$$\sum_k q_k(x_k) = x_l, \quad x \in [x], \tag{10}$$

$$[q_k] := \square\{q_k(x_k) \mid x_k \in [x_k]\}, \tag{11}$$

$$[s] := \sum_k [q_k]. \tag{12}$$

(i) If $0 \notin [r] := [x_l]-[s]$: then the box $[x]$ contains no feasible point.
(ii) If $0 \in [r]$ then

$$q_k(x_k) \in [r] \overset{\circ}{+} [q_k] \quad \textit{for all} \;\; k. \tag{13}$$

Proof. (10) implies

$$0 = x_l - \sum_k q_k(x_k) \in [x_l] - \sum_k [q_k] = [r]$$

and

$$q_k = x_l - \sum_k q_k + q_k \in [r] \overset{\circ}{+} [q_k]. \quad \square$$

The reduce process has two directions: In the *forward reduce* step, the enclosure for the right hand side is improved,

$$x_l \in [s] \cap [x_l].$$

In the *backward reduce* step, the enclosures for the variables in the left hand side are improved,

$$x_k \in q_k^{-1}([r] \overset{\circ}{+} [q_k]) \cap [x_k].$$

For simple functions q_k such as they occur in the predefined NOP elements, exact ranges are easy to compute. For more complicated functions q_k, enclosures for the ranges can be found using interval arithmetic; note that the proposition remains valid (but becomes less useful) when overestimating enclosures for (11) are used in place of the q_k.

If some intersection is empty, the box can be discarded. Of course, it may (and often does) happen that the intersection does not lead to a strict improvement of the box.

Mathematically equivalent but computationally slower forms of the above reduction process have been used in the UNICALC branch and bound zero finder [1] under the name of *subdefinite programming*, and in [2, 4, 11, 15, 16, 17, 26, 42] under the name of *interval constraint propagation*, used in a branch and bound context (for zeros of nonlinear systems) by VAN HENTENRYCK et al. [43].

6. Examples and test results

We first give a simple illustration for the working of **feas** and **reduce**. The NOP file for minimizing the well-known Rosenbrock function

$$f(x) = (10x_1^2 - 10x_2)^2 + (x_1 - 1)^2$$

in the box $x_1, x_2 \in [-2, 8]$ is

```
! Rosenbrock function
min dim4
bnd 1..2 in -2,8
! element list
qu4 1 2; 0 -10 10 0 x3
qu2 3 1; 0 1 x4
```

(cf. [35]; the last two lines contain indices and coefficients specifying separable quadratic equations). Expanded to mathematical notation, the NOP file requests the minimization of x_4 subject to the constraints

$$x_3 = 0x_1 - 10x_2 + 10x_1^2 + 0x_2^2 = 10x_1^2 - 10x_2,$$

and

$$x_4 = (x_3 - 0)^2 + (x_1 - 1)^2 = x_3^2 + (x_1 - 1)^2.$$

The initially missing bound constraints for x_3 and x_4 are taken in the program to be $[-10^9, 10^9]$. The absolutely smallest point of the initial box is $(0, 0, 0, 0)$, and is modified by **feas** to the feasible point $(0, 0, 0, 1)$. This gives an objective function value of $x_4 = 1$ and hence an improved bound $[x_4] = [-10^9, 1]$. If we now apply **reduce** we get from the first constraint

$$\begin{aligned}[x_3] &= (10[-2,8]^2 - 10[-2,8]) \cap [-10^9, 10^9]\\ &= [-80, 660] \cap [-10^9, 10^9] = [-80, 660]\end{aligned}$$

(forward reduce; backward reduce gives nothing new), and from the second constraint

$$\begin{aligned}[x_4] &= ([-80, 660]^2 + ([-2, 8] - 1)^2) \cap [-10^9, 1]\\ &= ([0, 435600] + [0, 49]) \cap [-10^9, 1]\\ &= [0, 435649] \cap [-10^9, 1] = [0, 1]\end{aligned}$$

(forward reduce) and then, with $[x]_+ = [\max(\underline{x}, 0), \overline{x}]$,

$$[x_3] = \sqrt{([0, 1] - [0, 49])_+} \cap [-80, 660] = [0, 1] \cap [-80, 660] = [0, 1],$$

$$\begin{aligned}[x_1] &= \left(1 + \sqrt{([0, 1] - [0, 435649])_+}\right) \cap [-2, 8]\\ &= [1, 2] \cap [-2, 8] = [1, 2]\end{aligned}$$

(backward reduce). The next pass of reduce still gives a nontrivial backward reduce for the first constraint:

$$[x_2] = (10[1, 2]^2 - [0, 1])/10 \cap [-2, 8]$$

$$= [0.9,4] \cap [-2,8] = [0.9,4].$$

Thus the initial box is reduced to $([1,2],[0.9,4],[0,1],[0,1])$. Note that we'd have got precisely the same results if we'd have optimized Rosenbrock's function with much looser bound constraints for x_1 and x_2!

For the behavior of the algorithm, a typical example is Problem 3 from Chapter 4 of FLOUDAS & PARDALOS [10]:

$$\begin{aligned} \min\ & x_1^{0.6} + x_2^{0.6} - 6x_1 - 4u_1 + 3u_2 \\ \text{s.t.}\ & x_2 - 3x_1 - 3u_1 = 0 \\ & x_1 + 2u_1 \le 4 \\ & x_2 + 2u_2 \le 4 \\ & x_1 \le 3 \\ & u_2 \le 1 \\ & x_1, x_2, u_1, u_2 \ge 0 \end{aligned}$$

```
min dim6
bnd 1 in 0,3
bnd 2 3 >= 0
bnd 4 in 0,1
pow 1 2; 0.6 x5
lin 5 1 3 4; 1 -6 -4 3 x6
lin 1 3; 3 3 x2
lin 1 3; 1 2 <= 4
lin 2 4; 1 2 <= 4
```

The output of our program (with a little more than minimal verbosity) looks as follows:

```
***********************************
initial FEAS found feasible point, f=  0.
 0.  0.  0.  0.  0.  0.  0.  0.
FEAS found a better point with f= -.716202736
FEAS found a better point with f= -4.02154303
     ...
       (some lines cancelled in output)
     ...
FEAS found a better point with f= -4.51420116
FEAS found a better point with f= -4.51420164
TUNNEL used 9 function calls
TUNNEL failed to find a better point
first narrow box discarded at nbox= 263
FEAS found a better point with f= -4.51420212
best point:
 1.33333302  4.  4.17232513E-07  7.94728621E-08
 3.48579812 -4.51420212  1.33333385  4.
first  wide  box discarded at nbox= 264
lower bound for minimum: -4.51420259
upper bound for minimum: -4.51420212
lower bound remained fixed since nbox= 263
best point defined at nbox= 264
264 boxes, 524 reduce calls, 277 f-values, 126 stack pos.
**************************************************
```

The first line says that in **presolve**, **feas** found a feasible point, given in line 2. Whenever better feasible points are found, their source (**feas** or **tunnel**) and their function values are displayed. Whenever **tunnel** is called, the effort spent on it is recorded, and whether it actually improved the best point. Here, this is not the case; **tunnel** tends to be efficient mainly in problems where **feas** performs poorly. Since without second order techniques, many narrow boxes may be generated, we drop boxes all of whose sides have length less than one millions of the box size after presolve. The first narrow box box dropped is monitored. When this occurs, the lowest lower bound f_{thresh} of function values over the set of dropped boxes cannot be superseded by later optimization steps, and wider boxes above this threshold are also discarded (again, when it occurs for the first time, it is recorded in the output). When the stack is empty, f_{thresh} is also taken as the final lower bound for the minimum; the upper bound is the best function value found. Finally, as a measure of performance, we record the total number of boxes processed, the total number of reduce calls, the total number of function evaluations in **feas** and **tunnel** steps, and the maximal number of boxes in the stack.

In Table I, we list the test problems (from three well-known collections [10, 19, 44] of global optimization test problems) that successfully passed our NOP reader. The first column contains the name of the problem, the second the types of constraints present (B bounds, L linear inequality, LE linear equality, Q quadratic inequality). The next two columns give the original dimension of the problem and its dimension in the NOP format. The final column indicates the extent to which GLOPT was able to solve the problems (+ successful global optimization with a narrow bound on the objective function value, $\circ$ good feasible points found but some wide boxes remained unexplored, $-$ no feasible point was found though the feasible region was nonempty). The number of boxes treated was limited to 10 000.

A number of problems were not solved to optimality since in our present implementation we neither incorporated local optimization steps nor higher order information or techniques from linear algebra. It is well-known that these are needed for high efficiency. The main purpose of the present study was to see how far one can go with the new zero-order techniques introduced here. For many problems these were already sufficient, and with not too many boxes processed.

The main source of inefficiency for the unsolved problems is the fact that one cannot yet combine simultaneous information from several constraints. In some problems, the failure was due to the fact that the reduction formulas for user-defined elements are so far implemented only in the forward mode, and with some overestimation due to the use of interval arithmetic. We are looking at ways to overcome these inefficiencies. Improving the reduction formulas for user-defined elements

Table I. Summary of test results

Name	constraint	dim.	dim. NOP	status
Six Hump Camel Back	B	2	3	+
Rastrigin	B	2	3	+
Griewank 2	B	2	3	+
Matyas	B	2	3	+
Rosenbrock	B	2	4	+
Simpl. Rosenbrock	B	2	4	+
Trecani	B	2	4	+
Booth	B	2	5	+
Goldstein-Price	B	2	5	$\circ$
Price	B	2	6	+
Schwefel 3.2	B	3	6	+
Schwefel 3.1	B	3	7	+
Schwefel 1.2	B	4	9	+
Powell	B	4	11	+
Kowalik	B	4	16	+
Schwefel 3.7	B	5	6	+
Griewank 10	B	10	13	+
Floudas/Pardalos 2T1	L,B	2	8	$\circ$
Floudas/Pardalos 2T2	L,B	6	9	+
Floudas/Pardalos 2T3	L,B	13	17	+
Floudas/Pardalos 2T4	L	6	8	+
Floudas/Pardalos 2T5	L,B	10	12	$\circ$
Floudas/Pardalos 2T6	L,B	10	12	+
Floudas/Pardalos 2T7.1	L	20	21	$\circ$
Floudas/Pardalos 2T7.2	L	20	21	$\circ$
Floudas/Pardalos 2T7.3	L	20	21	$\circ$
Floudas/Pardalos 2T7.4	L	20	21	$\circ$
Floudas/Pardalos 2T7.5	L	20	21	−
Floudas/Pardalos 2T8	LE,L	24	25	−
Floudas/Pardalos 2T9	LE,L	10	11	$\circ$
Floudas/Pardalos 2T10	LE,L	20	23	$\circ$
Floudas/Pardalos 3T3	Q,B	6	9	$\circ$
Floudas/Pardalos 4T3	LE,B	4	6	+
Floudas/Pardalos 4T4	LE,B	4	7	+
Floudas/Pardalos 4T5	LE,B	6	9	+

is not too difficult, but it seriously affects only a very small number of our test problems. We are currently working on an implementation of underestimation techniques and second order techniques (related to

NEUMAIER [34]) that eliminate the problem of not using simultaneous information, thus overcoming the main inefficiency.

Acknowledgment

Part of this work has been done with support by the European External Research Program (Contract No. AU-043) by DIGITAL Equipment Corp., Austria.

Thanks also go to a referee for pointing out the relations of our box reduction technique to interval constraint propagation.

References

1. A. B. Babichev, O. B. Kadyrova, T. P. Kashevarova, A. S. Leshchenko, and A. L. Semenov, UniCalc, a novel approach to solving systems of algebraic equations, Interval Computations 3 (1993),29–47.
2. F. Benhamou and W. J. Older, Applying interval arithmetic to real, integer, and boolean constraints, J. of Math. Mech., 1995.
3. S. Berner, New results on verified global optimization, submitted 1995.
4. H. M. Chen and M. H. van Emden, Adding interval constraints to the Moore–Skelboe global optimization algorithm, pp. 54–57 in: V. Kreinovich (ed.) Extended Abstracts of APIC'95, International Workshop on Applications of Interval Computations, Reliable Computing (Supplement), 1995.
5. T. Csendes and D. Ratz, Subdivision direction selection in interval methods for global optimization, SIAM J. Numer. Anal., to appear.
6. L. Davis (ed.), Handbook of genetic algorithms, van Nostrand Reinhold, New York 1991.
7. J.E. Dennis, jr., D.M. Gay, and R.E. Welsch, Algorithm 573. NL2SOL – An adaptive nonlinear least-squares algorithm, ACM Trans. Math. Softw. 7 (1981), 369-383.
8. L.C.W. Dixon and G.P. Szegö, Towards global optimization, Elsevier, New York 1975.
9. T. G. W. Epperly, Global optimization of nonconvex nonlinear programs using parallel branch and bound, PH. D. Thesis, Dept. of Chem. Eng., University of Wisconsin – Madison, 1995.
10. C. A. Floudas and P.M. Pardalos, A collection of test problems for constrained global optimization algorithms, Lecture Notes Comp. Sci. 455, Springer, Berlin 1990.
11. G. D. Hager, Solving large systems of nonlinear constraints with application to data modeling, Interval Computations 3 (1993),169–200.
12. E. Hansen, Global optimization using interval analysis, Dekker, New York 1992.
13. J. Holland, Genetic algorithms and the optimal allocation of trials, SIAM J. Computing 2 (1973), 88-105.
14. R. Horst and P.M. Pardalos (eds.), Handbook of global optimization, Kluwer 1995.
15. E. Hyvönen, Constraint reasoning based on interval arithmetic, In N. S. Sridharan (ed.), IJCAI-89 Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, 1989.
16. E. Hyvönen, Global consistency in interval constraint satisfaction, In B. Mayoh (ed.), Scandinavian Conference on Artificial Intelligence - 91. 'Odin's Ravens'. Proceedings of the SCAI '91, 1991.

17. E. Hyvönen and S. De Pascale, Interval computations on the spreadsheet. pp. 169–209 in: R. B. Kearfott and V. Kreinovich (eds.), Applications of Interval Computations, Applied Optimization, Kluwer, Dordrecht 1996.
18. C. Jansson, On self-validating methods for optimization problems, pp. 381-438 in: Topics in validated computations, (J. Herzberger, ed.): Elsevier Science B.V., 1994.
19. C. Jansson and O. Knüppel, A global minimization method: The multi-dimensional case, Preprint, 1992.
20. C. Jansson and O. Knüppel, Branch and bound algorithm for bound constrained optimization problems without derivatives, J. Global Optim. 7 (1995),297-333.
21. R.B. Kearfott, M. Novoa and Chenyi Hu, A review of preconditioners for the interval Gauss–Seidel method, Interval Computations 1 (1991), 59–85.
22. R. B. Kearfott, Decomposition of arithmetic expressions to improve the behavior of interval iteration for nonlinear systems, Computing, 47 (1991), 169–191.
23. R.B. Kearfott, A review of techniques in the verified solution of constrained global optimization problems, to appear.
24. S. Kirkpatrick, C.D. Geddat, jr., and M.P. Vecchi, Optimization by simulated annealing, Science 220 (1983), 671-680.
25. A.V. Levy, and A. Montalvo, The tunneling algorithm for the global minimization of functions, SIAM J. Sci. Stat. Comput. 6 (1985), 15-29.
26. W. A. Lodwick, Constraint propagation, relational arithmetic in AI systems and mathematical programs, Ann. Oper. Res. 21 (1989), 143–148.
27. C.D. Maranas and C.A. Floudas, Global minimum potential energy conformations of small molecules, J. Global Optim. 4 (1994), 135-170.
28. C.D. Maranas and Ch. A. Floudas, Finding all solutions of nonlinearly constrained systems of equations, J. Global Optim. 7 (1995), 143-182.
29. J. Mockus, Bayesian approach to global optimization, Kluwer, Dordrecht 1989.
30. R.E. Moore, Methods and applications of interval analysis, SIAM, Philadelphia 1979.
31. G.L. Nemhauser and L.A. Wolsey, Integer Programming, Chapter VI (pp. 447-527) in: Optimization (G.L. Nemhauser et al., eds.), Handbooks in Operations Research and Management Science, Vol. 1, North Holland, Amsterdam 1989.
32. A. Neumaier, The enclosure of solutions of parameter-dependent systems of equations, pp. 269-286 in: Reliability in Computing (R.E. Moore, ed.), Acad. Press, San Diego 1988.
33. A. Neumaier, Interval methods for systems of equations, Cambridge Univ. Press, Cambridge 1990.
34. A. Neumaier, Second-order sufficient optimality conditions for local and global nonlinear programming, J. Global Optimization, to appear.
35. A. Neumaier, NOP – a compact input format for nonlinear optimization problems, submitted to J. Global Optim..
36. P.M Pardalos and J.B. Rosen, Constrained global optimization: Algorithms and applications, Lecture Notes in Computer Science 268, Springer, Berlin 1987.
37. D. Ratz, On branching rules in second-order branch-and-bound methods for global optimization, in: G. Alefeld et al. (eds.), Scientific computation and validation, Akademie-Verlag, Berlin 1996.
38. D. Ratz and T. Csendes, On the selection of subdivision directions in interval branch-and-bound methods for global optimization, J. Global. Optim., to appear.
39. C.A. Schnepper and M.A. Stadtherr, Application of a parallel interval Newton/generalized bisection algorithm to equation-based chemical process flowsheeting, Interval Computations 4 (1994), 40-64.

40. X. Shi, Intermediate Expression Preconditioning and Verification for Rigorous Solution of Nonlinear Systems, PhD thesis, University of Southwestern Louisiana, Department of Mathematics, August 1995.
41. A. Törn and A. Žilinskas, Global optimization, Lecture Notes in Computer Science 350, Springer, Berlin 1989.
42. P. Van Hentenryck, Constraint Satisfaction in Logic Programming, MIT Press, Cambridge, MA, 1989.
43. P. Van Hentenryck, D. McAllester, and D. Kapur, Solving polynomial systems using a branch and prune approach, Technical Report CS-95-01, Dept. of Comp. Sci., Brown University, 1995.
44. G. Walster, E. Hansen and S. Sengupta, Test results for a global optimization algorithm, pp.272-287 in: Numerical optimization 1984 (P.T. Boggs et al., eds), SIAM, Philadelphia 1985.