Global Optimization Using the DIRECT Algorithm in Matlab

Mattias Björkman and Kenneth Holmström

Applied Optimization and Modeling Group (**TOM**) [2]

Center for Mathematical Modeling
Department of Mathematics and Physics
Mälardalen University
P.O. Box 883, S-721 23 Västerås, Sweden

Research Report in MATHEMATICS / APPLIED MATHEMATICS
Technical Report IMa-TOM-1998-12

June 1, 1999

Abstract

In this paper we will discuss the efficiency and implementation details of an algorithm for finding the global minimum of a multivariate function subject to simple bounds on the variables. The algorithm, DIRECT, developed by D. R. Jones, C. D. Perttunen and B. E. Stuckman is a modification of the standard Lipschitzian approach that eliminates the need to specify a Lipschitz constant.

We have implemented the DIRECT algorithm in Matlab and the efficiency of our implementation is analyzed by comparing it to the result of Jones's implementation on nine standard test problems for global optimization. In fifteen out of eighteen runs the results is to the favor of our implementation. For some test problems the differences in the number of function evaluations needed for the algorithm to converge are small but for others the differences are great enough to be worth a discussion.

Our code is integrated in the NLPLIB TB Toolbox as part of the optimization environment TOMLAB. All tests are performed in TOMLAB.

1 Introduction

TOMLAB, [2, 4], developed by the applied optimization and modeling group (TOM) at Mälardalen University, is an open Matlab environment for research and teaching in optimization. TOMLAB is based on NLPLIB TB, [3], a Matlab toolbox for nonlinear programming and parameter estimation, and OPERA TB, [5], a Matlab toolbox for linear and discrete optimization. Although TOMLAB includes more than 65 different algorithms and graphical utilities there has until recently been no routine to handle global optimization problems. Therefore the DIRECT algorithm focused our interest.

DIRECT is an algorithm, presented by D. R. Jones et al. in [7], for finding the global minimum of a multivariate function subject to simple bounds. The algorithm is a modification of the standard Lipschitzian approach that eliminates the need to specify a Lipschitz constant. This is done by carrying out simultaneous searches using all possible constants from zero to infinity. In [7], Jones et al. introduces a different way of looking at the Lipschitz constant. In particular, the Lipschitz constant is viewed as a weighting parameter that indicate how much emphasis to place on global versus local search. In standard Lipschitzian methods, this constant is usually large because it must equal or exceed the maximum rate of change of the objective function. As a result, these methods place a high emphasis on global search which leads to slow convergence. In contrast, the DIRECT algorithm carries out simultaneous searches using all possible constants, and therefore operates on both the global and local level. Once the global part of the algorithm finds the basin of convergence of the optimum, the local part quickly and automatically exploits it. This accounts for the fast convergence of the DIRECT algorithm.

To avoid any misunderstandings we will now make clear for the reader what we mean by the names of algorithms and implementations which will be used throughout this paper. By DIRECT we mean the algorithm described in [7]. When we talk in comparative terms we will by DIRECT mean the implementation made by Jones, Perttunen and Stuckman. Our implementation of the DIRECT algorithm (including modifications) will be referred to as *glbSolve* which also is the name of the corresponding Matlab routine included in the NLPLIB TB.

This paper is organized as follows. We begin in Section 2 by presenting some of the key ideas behind the DIRECT algorithm. We will also give a very formal description of it which follows the implementation in *glbSolve*. One important phase in the algorithm is to find the extreme points on the lower convex hull of a set of points in the plane. A formal description of the technique we use for that problem and a presentation of other implementation details is also given in Section 2. In Section 3 we define the test problems used in the analysis of the implementation efficiency. In Section 4 we present the results of the comparison of *glbSolve* versus DIRECT. We also present a graphical illustration of the search behavior. Finally, Section 5 summarizes the results from our point of view.

2 The Algorithm

We will in the beginning of this section discuss some of the key ideas of the DIRECT algorithm. We see no point in giving a complete description and motivation for those ideas, instead we refer to [7] where the reader could get all that information presented in a good way. Some details in our implementation which is not described in [7] will be pointed out. Then we end up this section by giving a formal description of the algorithm following our implementation in *glbSolve*.

The mathematical formulation of the problem which DIRECT is aimed to solve is given by

$$\begin{array}{llcccc} \min\limits_{x} & f(x) & & & & \\ s/t & x_L & \leq & x & \leq & x_U \end{array},$$

where $f \in \mathbb{R}$ is continuous and $x, x_L, x_U \in \mathbb{R}^n$.

The first step in the DIRECT algorithm is to transform the search space to be the unit hypercube. The function is then sampled (its value is computed) at the centerpoint of this cube. Computing the function value at the centerpoint instead of doing it at the vertices has an advantage when dealing with problems in higher dimensions. The hypercube is then divided into smaller hyperrectangles whose centerpoints are sampled. Instead of using a Lipschitz constant when determining which rectangles to sample next, DIRECT identifies a set of *potentially optimal* rectangles in each iteration. All those *potentially optimal* rectangles are further divided into smaller rectangles whose centerpoints are sampled. When no Lipschitz constant is used, there is no natural way of defining convergence (except when the optimal function value is known as in the test problems). Instead, the procedure described above is performed for a given number of iterations.

The problem of finding the extreme points on the lower convex hull of a set of points in the plane is introduced as a subproblem when to determine the set of all *potentially optimal* rectangles. In our implementation, *glbSolve*, those points are identified by use of the *glbSolve* subfunction *conhull*. As a result of the technique we use in our implementation of DIRECT, *conhull* will assume that the points are sorted by increasing abscissa. The implementation of *conhull* is based on the algorithm GRAHAMHULL in [8, page 108] with the modifications proposed on page 109. The initial step in GRAHAMHULL is to sort the points by increasing abscissas. As mentioned above this is already done so this initial step is skipped. We give a formal description of the *conhull* algorithm below.

conhull

The points (x_i, y_i), $i = 1, 2, 3, ..., m$ are given with $x_1 \leq x_2 \leq ... \leq x_m$.
Set $h = (1, 2, ..., m)$.
if $m \geq 3$ **then**
  Set $START = 1$, $v = START$, $w = m$ and $flag = 0$.

while $next(v) \neq START$ **or** $flag = 0$ **do**
  if $next(v) = w$ **then**
    Set $flag = 0$.
  end if
  Set $a = v$, $b = next(v)$ and $c = next(next(v))$.
  Set $A = \begin{pmatrix} x_a & y_a & 1 \\ x_b & y_b & 1 \\ x_c & y_c & 1 \end{pmatrix}$.
  if $\det(A) \geq 0$ **then**
    Set $leftturn = 0$.
  else
    Set $leftturn = 1$.
  end if
  if $leftturn$ **then**
    Set $v = next(v)$.
  else
    Set $j = next(v)$.
    Set $x = (x_1, x_2, ..., x_{j-1}, x_{j+1}, ..., x_m)$, $y = (y_1, y_2, ..., y_{j-1}, y_{j+1}, ..., y_m)$ and
      $h = (h_1, h_2, ..., h_{j-1}, h_{j+1}, ..., h_m)$.
    Set $m = m - 1$, $w = w - 1$ and $v = pred(v)$.
  end if
end while
end if

Now, h is an index vector containing the indices to the points which lies on the convex hull. The function $next(v)$ returns $v + 1$ if $v < m$ and 1 if $v = m$. The function $pred(v)$ returns $v - 1$ if $v > 1$ and m if $v = 1$.

The version of the DIRECT algorithm implemented in *glbSolve* is formally described below. Instead of using the tree structure proposed in [7] for storing the information of each rectangle we use a straightforward matrix/vector technique. The notation used will be explained in Table 1.

glbSolve

Set the global/local search weight parameter ϵ.
Set $C_{i1} = \frac{1}{2}$ and $L_{i1} = \frac{1}{2}$, $i = 1, 2, 3, ..., n$.
Set $F_1 = f(x)$, where $x_i = x_{L_i} + C_{i1}(x_{U_i} - x_{L_i})$, $i = 1, 2, 3, ..., n$.
Set $D_1 = \sqrt{\sum_{k=1}^{n} L_k^2}$.
Set $f_{min} = F_1$ and $i_{min} = 1$.
for $t = 1, 2, 3, ..., T$ **do**

Table 1: Notation used in the *glbSolve* algorithm description.

Parameter	Explanation
C	Matrix with all rectangle centerpoints.
D	Vector with distances from centerpoint to the vertices.
F	Vector with function values.
I	Set of dimensions with maximum side length for the current rectangle.
L	Matrix with all rectangle side lengths in each dimension.
S	Index set of *potentially optimal* rectangles.
T	The number of iterations to be performed.
f_{min}	The current minimum function value.
i_{min}	Rectangle index.
ϵ	Global/local search weight parameter, by default $\epsilon = 10^{-4}$.
δ	New side length in the current divided dimension.

Set $\hat{S} = \left\{ j : D_j \geq D_{i_{min}} \wedge F_j = \min\limits_i \{F_i : D_i = D_j\} \right\}$.
Define α and β by letting the line $y = \alpha x' + \beta$ pass through the points $(D_{i_{min}}, F_{i_{min}})$ and $\left(\max\limits_j (D_j), \min\limits_i \left(F_i : D_i = \max\limits_j (D_j) \right) \right)$.
Let $\tilde{S}$ be the set of all rectangles $j \in \hat{S}$ fulfilling $F_j \leq \alpha D_j + \beta + 10^{-12}$.
Let S be the set of all rectangles in $\tilde{S}$ which lies on the convex hull defined by the points (D_j, F_j), $j \in \tilde{S}$.
while $S \neq \emptyset$ **do**
  Select j as the first element in S.
  Set $S = S \setminus \{j\}$.
  Let I be the set of dimensions with maximum rectangle side length, i.e. $I = \left\{ i : D_{ij} = \max\limits_k (D_{kj}) \right\}$.
  Let δ equal two-thirds of this maximum side length, i.e. $\delta = \frac{2}{3} \max\limits_k (D_{kj})$.
  for all $i \in I$ **do**
    Set $c_k = C_{kj}$, $k = 1, 2, 3, ..., n$.
    Set $\hat{c} = c + \delta e_i$ and $\tilde{c} = c - \delta e_i$, where e_i is the ith unit vector.
    Compute $\hat{f} = f(\hat{x})$ and $\tilde{f} = f(\tilde{x})$ where $\hat{x}_k = x_{L_k} + \hat{c}_k \left(x_{U_k} - x_{L_k} \right)$ and $\tilde{x}_k = x_{L_k} + \tilde{c}_k \left(x_{U_k} - x_{L_k} \right)$.
    Set $w_i = \min(\hat{f}, \tilde{f})$.
    Set $C = \begin{pmatrix} C & \hat{c} & \tilde{c} \end{pmatrix}$ and $F = \begin{pmatrix} F & \hat{f} & \tilde{f} \end{pmatrix}$.
  end for
  while $I \neq \emptyset$ **do**
    Select the dimension $i \in I$ with the lowest value of w_i and set $I = I \setminus \{i\}$.

Set $L_{ij} = \frac{1}{2}\delta$.
Let $\hat{j}$ and $\tilde{j}$ be the indices that corresponds to the points $\hat{c}$ and $\tilde{c}$ above.
Set $L_{k\hat{j}} = L_{kj}$ and $L_{k\tilde{j}} = L_{kj}$, $k = 1, 2, 3, ..., n$.
Set $D_j = \sqrt{\sum_{k=1}^{n} L_{kj}^2}$.
Set $D_{\hat{j}} = D_j$ and $D_{\tilde{j}} = D_j$.
end while
end while
Set $f_{min} = \min_j(F_j)$.
Set $i_{min} = \text{argmin}\left(\frac{F_j - f_{min} + E}{D_j}\right)$, where $E = \max\left(\epsilon|f_{min}|, 10^{-8}\right)$.
end for

3 The Test Problems

In this section, we will give a complete description of the test problems used in the comparisons in Section 4. We use the same problem names and abbreviations as in [7]. First of all, we give in Table 2 a compact description of the test problems. This compact description includes information about the abbreviation used, the problem dimension and the number of local and global minima.

Table 2: Compact description of the test problems.

Problem	Abbreviation	Number of dimensions	Number of local minima	Number of global minima
Shekel 5	S5	4	5	1
Shekel 7	S7	4	7	1
Shekel 10	S10	4	10	1
Hartman 3	H3	3	4	1
Hartman 6	H6	6	4	1
Branin RCOS	BR	2	3	3
Goldstein and Price	GP	2	4	1
Six-Hump Camel	C6	2	6	2
Two-Dimensional Schubert	SHU	2	760	18

The problem definitions of problem S5, S7, S10, H3, H6 and GP are taken from [1], problem BR from [6, page 468] and problem C6 and SHU from [9]. All information about the optimal function values, denoted f_{global}, were kindly supplied by D. R. Jones.

- The problems Schekel 5, Schekel 7 and Schekel 10 is defined by

$$\begin{array}{ll} \min\limits_x & f(x) = -\sum\limits_{i=1}^{m} \frac{1}{(x-a_i)^T(x-a_i)+c_i} \\ s/t & 0 \ \le \ x_j \ \le 10 \quad j = 1, 2, ..., n \end{array},$$

where $n = 4$, a_i is the ith row in A and c_i is the ith element in c. A and c is defined by

$$A = \begin{pmatrix} 4 & 4 & 4 & 4 \\ 1 & 1 & 1 & 1 \\ 8 & 8 & 8 & 8 \\ 6 & 6 & 6 & 6 \\ 3 & 7 & 3 & 7 \\ 2 & 9 & 2 & 9 \\ 5 & 5 & 3 & 3 \\ 8 & 1 & 8 & 1 \\ 6 & 2 & 6 & 2 \\ 7 & 3.6 & 7 & 3.6 \end{pmatrix}, c = \begin{pmatrix} 0.1 \\ 0.2 \\ 0.2 \\ 0.4 \\ 0.4 \\ 0.6 \\ 0.3 \\ 0.7 \\ 0.5 \\ 0.5 \end{pmatrix}.$$

Consider the three cases with $m = 5, 7$ and 10.

$$\begin{array}{ll} m = 5: & f_{global} = -10.1531996790582 \\ m = 7: & f_{global} = -10.4029405668187 \\ m = 10: & f_{global} = -10.5364098166920 \end{array}$$

- The problems Hartman 3 and Hartman 6 is defined by

$$\begin{array}{ll} \min\limits_x & f(x) = -\sum\limits_{i=1}^{m} c_i \exp\left(-\sum\limits_{j=1}^{m} a_{ij}(x_j - p_{ij})^2\right) \\ s/t & 0 \ \le \ x_j \ \le 1 \quad j = 1, 2, ..., n \end{array}.$$

For Hartman 3, $n = 3$ and A, c and P is given by

$$A = \begin{pmatrix} 3 & 10 & 30 \\ 0.1 & 10 & 35 \\ 3 & 10 & 30 \\ 0.1 & 10 & 35 \end{pmatrix}, c = \begin{pmatrix} 1 \\ 1.2 \\ 3 \\ 3.2 \end{pmatrix}, P = \begin{pmatrix} 0.3689 & 0.1170 & 0.2673 \\ 0.4699 & 0.4387 & 0.7470 \\ 0.1091 & 0.8732 & 0.5547 \\ 0.03815 & 0.5743 & 0.8828 \end{pmatrix},$$

and for Hartman 6, $n = 6$ and A, c and P is given by

$$A = \begin{pmatrix} 10 & 3 & 17 & 3.5 & 1.7 & 8 \\ 0.05 & 10 & 17 & 0.1 & 8 & 14 \\ 3 & 3.5 & 1.7 & 10 & 17 & 8 \\ 17 & 8 & 0.05 & 10 & 0.1 & 14 \end{pmatrix}, c = \begin{pmatrix} 1 \\ 1.2 \\ 3 \\ 3.2 \end{pmatrix},$$

$$P = \begin{pmatrix} 0.1312 & 0.1696 & 0.5569 & 0.0124 & 0.8283 & 0.5886 \\ 0.2329 & 0.4135 & 0.8307 & 0.3736 & 0.1004 & 0.9991 \\ 0.2348 & 0.1451 & 0.3522 & 0.2883 & 0.3047 & 0.6650 \\ 0.4047 & 0.8828 & 0.8732 & 0.5743 & 0.1091 & 0.0381 \end{pmatrix}.$$

$$\begin{array}{l} n = 3: \;\; f_{global} = -3.86278214782076 \\ n = 6: \;\; f_{global} = -3.32236801141551 \end{array}$$

- The Branin RCOS problem is defined by

$$\begin{array}{ll} \min\limits_{x} & f(x) = \left(x_2 - \frac{5x_1^2}{4\pi^2} + \frac{5x_1}{\pi} - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos(x_1) + 10 \\ s/t & \begin{array}{rcccl} -5 & \leq & x_1 & \leq & 10 \\ 0 & \leq & x_2 & \leq & 15 \end{array} \end{array}.$$

$$f_{global} = 0.397887357729739$$

- The Goldstein and Price problem is defined by

$$\begin{array}{llcl} \min\limits_{x} & f(x) & = & \left[1 + (x_1 + x_2 + 1)^2 (19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)\right] \\ & & \times & \left[30 + (2x_1 - 3x_2)^2 (18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)\right] \\ s/t & & & -2 \;\leq\; x_j \;\leq 2 \quad j = 1, 2 \end{array}.$$

$$f_{global} = 3$$

- The Six-Hump Camel problem is defined by

$$\begin{array}{ll} \min\limits_{x} & f(x) = \left(4 - 2.1x_1^2 + \frac{1}{3}x_1^4\right)x_1^2 + x_1x_2 + (-4 + 4x_2^2)\,x_2^2 \\ s/t & \begin{array}{rcccl} -3 & \leq & x_1 & \leq & 3 \\ -2 & \leq & x_2 & \leq & 2 \end{array} \end{array}.$$

$$f_{global} = -1.0316284535$$

- Finally, the Two-Dimensional Schubert problem is defined by

$$\begin{array}{ll} \min\limits_{x} & f(x) = \left[\sum\limits_{i=1}^{5} i\cos\left[(i+1)x_1 + i\right]\right] \times \left[\sum\limits_{i=1}^{5} i\cos\left[(i+1)x_2 + i\right]\right] \\ s/t & -10 \;\leq\; x_j \;\leq 10 \quad j = 1, 2 \end{array}.$$

$$f_{global} = -186.730908831024$$

4 Numerical Results

In this section we will present some numerical experience that have been obtained with our implementation of DIRECT in *glbSolve*.

To illustrate the search behavior we have as in [7] chosen to present scatter plots for the Branin's problem after 16 and 45 iterations with 231 and 1017 function evaluations respectively, see Figure 1 and 2. Branin's function have three global optimum and the sampled points clearly cluster around them.

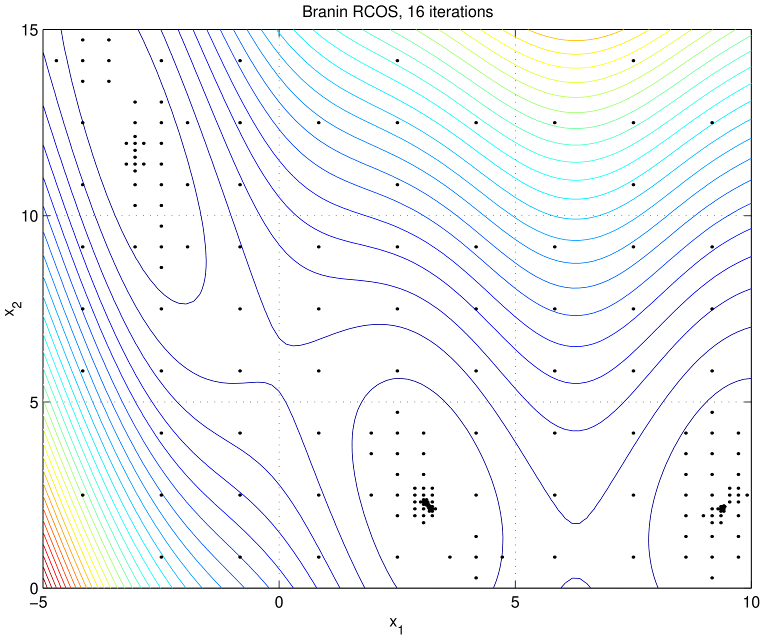

Figure 1: Scatter plot of Branin RCOS after 16 iterations and 231 function evaluations

In Table 3 we compare the efficiency of *glbSolve* versus DIRECT. The efficiency is measured in form of the number of function evaluations needed for convergence. We use the same definition of convergence as in [7] where convergence is defined in terms of percent error from the globally optimal function value. Let f_{global} denote the known optimal function value and let f_{min} denote the best function value at some point in the search, then the percent error is defined by

$$error = 100\frac{f_{min} - f_{global}}{|f_{global}|}.$$

In all runs, the parameter ϵ was set to 10^{-4}. This is the value of ϵ used in [7].

As you can see in table 3, *glbSolve* needs fewer function evaluations in 15 of the 18 runs. For the first six problems, the differences are small but that is not a fact for

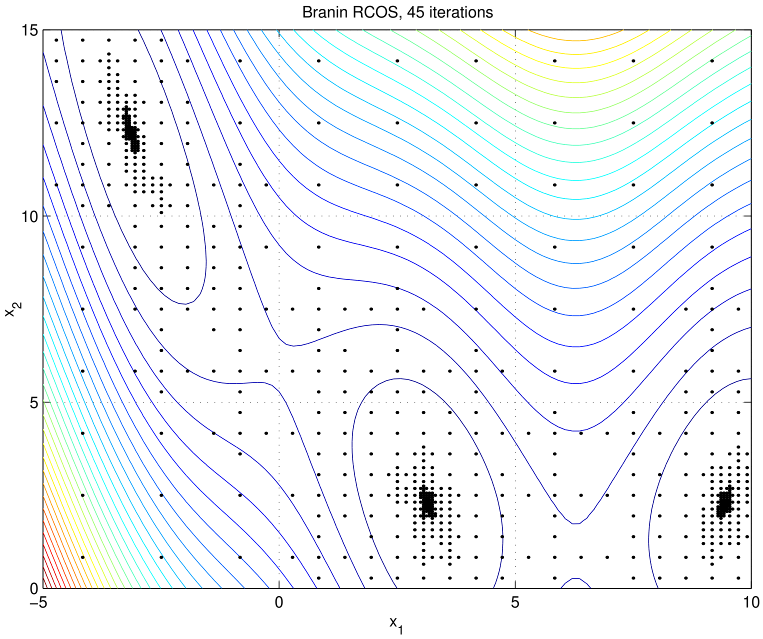

Figure 2: Scatter plot of Branin RCOS after 45 iterations and 1017 function evaluations

Table 3: Number of function evaluations, *glbSolve* vs. DIRECT

Routine		S5	S7	S10	H3	H6	BR	GP	C6	SHU
glbSolve	$error < 1\%$	100	94	94	70	198	65	83	77	3193
DIRECT	$error < 1\%$	103	97	97	83	213	63	101	113	2883
glbSolve	$error < 0.01\%$	153	143	143	178	529	165	167	146	3274
DIRECT	$error < 0.01\%$	155	145	145	199	571	195	191	285	2967

problem C6 and SHU. It is difficult to give a clear explanation of those great differences but for problem SHU with 760 local minima and 18 global minima, we think that the numerical tolerances used in the implementations play an important role. We think that it is important to mention that we have spend no time and no effort adjusting or tuning those numerical tolerances.

5 Conclusions

We have presented in this paper some details and some numerical results concerning the implementation of the DIRECT algorithm in Matlab. The Matlab environment gives access to several utility functions which makes the implementation rather straightforward.

Further, letting *glbSolve* be a part of the TOMLAB toolbox for nonlinear programming NLPLIB TB it could be used for many purposes.

The numerical results obtained are pleasant. In [7], DIRECT is shown to perform well compared to other algorithms. We compare our implementation to DIRECT and the results are to our favor in most of the runs.

References

[1] L. C. W. Dixon and G. P. Szegö. The global optimisation problem: An introduction. In L. Dixon and G Szego, editors, *Toward Global Optimization*, pages 1–15, New York, 1978. North-Holland Publishing Company.

[2] Kenneth Holmström. The TOMLAB Optimization Environment in Matlab. *Advanced Modeling and Optimization*, 1(1):47–69, 1999.

[3] Kenneth Holmström and Mattias Björkman. The TOMLAB NLPLIB Toolbox for Nonlinear Programming. *Advanced Modeling and Optimization*, 1(1):70–86, 1999.

[4] Kenneth Holmström, Mattias Björkman, and Erik Dotzauer. A User's Guide to the Optimization Environment TOMLAB 1.0. Technical Report IMa-TOM-1998-09, Department of Mathematics and Physics, Mälardalen University, Sweden, 1998. Revised April 20, 1999.

[5] Kenneth Holmström, Mattias Björkman, and Erik Dotzauer. The TOMLAB OPERA Toolbox for Linear and Discrete Optimization. *Advanced Modeling and Optimization*, 1(2):1–8, 1999.

[6] Reiner Horst and Panos M. Pardalos. *Handbook of Global Optimization*. Kluwer Academic Publishers, Dordrecht Boston London, 1995.

[7] D. R. Jones, C. D. Perttunen, and B. E. Stuckman. Lipschitzian optimization without the Lipschitz constant. *Journal of Optimization Theory and Applications*, 79(1):157–181, October 1993.

[8] Franco P. Preparata and Michael Ian Shamos. *Computational Geometry*. Springer-Verlag, New York, 1985.

[9] Yong Yao. Dynamic tunneling algorithm for global optimization. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(5):1222–1230, 1989.