WORKING PAPER
MANSHOLT GRADUATE SCHOOL

On the Goodness of Global Optimisation

Algoritms, an introduction into investigating algorithms

Eligius M.T. Hendrix

DISCUSSION PAPER No. 21
2005

Mansholt Graduate School of Social Sciences

Hollandseweg 1, 6706 KN Wageningen,
The Netherlands
Phone: +31 317 48 41 26
Fax: +31 317 48 47 63
Internet: http://www.mansholt.wur.nl/
e-mail: office.mansholt@wur.nl

Working Papers are interim reports on work of Mansholt Graduate School of Social Sciences (MGS) and have received only limited reviews[1]. Each paper is refereed by one member of the Editorial Board and one member outside the board. Views or opinions expressed in them do not necessarily represent those of the Mansholt Graduate School.

The Mansholt Graduate School's researchers are based in three departments: 'Social Sciences', 'Environmental Sciences' and 'Agrotechnology and Food sciences' and two institutes: 'LEI, Agricultural Economics Research Institute' and 'Alterra, Research Institute for the Green World'. In total Mansholt Graduate School comprises about 250 researchers.

Mansholt Graduate School is specialised in social scientific analyses of the rural areas and the agri- and food chains. The Graduate School is known for its disciplinary and interdisciplinary work on theoretical and empirical issues concerning the transformation of agriculture, rural areas and chains towards multifunctionality and sustainability.

Comments on the Working Papers are welcome and should be addressed directly to the author(s).

Eligius M.T. Hendrix[a]

[a] Operations Research and Logistics Group, Wageningen University
Hollandseweg 1, 6706 KN Wageningen, the Netherlands

[1] Working papers may have been submitted to other journals and have entered a journal's review process. Should the journal decide to publish the article the paper no longer will have the status of a Mansholt Working Paper and will be withdrawn from the Mansholt Graduate School's website. From then on a link will be made to the journal in question referring to the published work and its proper citation.

On the Goodness of Global Optimisation Algorithms, an introduction into investigating algorithms

Eligius M.T. Hendrix

November 9, 2005

Abstract

An early introductory text on Global Optimisation (GO), Törn and Zilinskas (1989), goes further than mathematical correctness in giving the reader an intuitive idea about concepts in GO. This paper extends this spirit by introducing students and researchers to the concepts of Global Optimisation (GO) algorithms. The goal is to learn to read and interpret optimisation algorithms and to analyse their goodness. Before going deeper into mathematical analysis, it is good for students to get a flavour of the difficulty by letting them experiment with simple algorithms that can be followed by hand or spreadsheet calculations. Two simple one-dimensional examples are introduced and several simple NLP and GO algorithms are elaborated. This is followed by some lessons that can be learned from investigating the algorithms systematically.

1 Effectiveness and efficiency of algorithms

In this paper, several criteria are discussed to measure effectiveness and efficiency of algorithms. Moreover, examples are given of basic algorithms that are analysed. This gives an opportunity to introduce in an illustrative way GO concepts such as *region of attraction*, *level set*, *probability of success* and *Performance Graph*. To investigate optimisation algorithms, we start with a definition. An algorithm is a description of steps, preferably implemented into a computer program, which finds an approximation of an optimum point. The aims can be several: reach a local optimum point, reach a global optimum point, find all global optimum points, reach all global and local optimum points. In general, an algorithm generates a series of points x_k that approximate an (or the or all) optimum point. According to the generic description of Törn and Zilinskas (1989):

$$x_{k+1} = Alg(x_k, x_{k-1}, ..., x_0, \xi) \tag{1}$$

where ξ is a random variable and index k is the iteration counter. This represents the idea that a next point x_{k+1} is generated based on the information in all

former points $x_k, x_{k-1}, ..., x_0$ (x_0 usually being the starting point) and possibly some random effect. This leads to three types of algorithms discussed here:

- Nonlinear optimisation algorithms, that from a starting point try to capture the "nearest" local minimum point.
- Deterministic GO methods which guarantee to approach the global optimum and require a certain mathematical structure.
- Stochastic GO methods based on the random generation of feasible trial points and nonlinear local optimization procedures.

We will consider several examples illustrativing two questions to address to investigate the quality of algorithms (see Baritompa and Hendrix (2005)).

- Effectiveness: does the algorithm find what we want?
- Efficiency: what are the computational costs?

Several measurable performance indicators can be defined for these criteria.

1.1 Effectiveness

Consider minimisation algorithms. Focusing on effectiveness, there are several targets a user may have:

1. To discover all global minimum points. This of course can only be realised when the number of global minimum points is finite.
2. To detect at least one global optimal point.
3. To find a solution with a function value as low as possible.

The first and second targets are typical satisfaction targets; was the search successful or not? What are good **measures of success**? In the literature, convergence is often used i.e. $x_k \to x^*$, where x^* is one of the minimum points. Alternatively one observes $f(x_k) \to f(x^*)$. In tests and analyses, to make results comparable, one should be explicit in the definitions of success. We need not only specify ϵ and/or δ such that

$$\|x_k - x^*\| < \epsilon \text{ and/or } \quad f(x_k) < f(x^*) + \delta \tag{2}$$

but also specify whether success means that there is an index K such that (3) is true for all $k > K$. Alternatively, success may mean that a record $min_k f(x_k)$ has reached level $f(x^*) + \delta$. Whether the algorithm is effective also depends on the its stochastic nature. When we are dealing with stochastic algorithms, effectiveness can be expressed as the probability that a success has been reached. In analysis, this probability can be derived from sufficient assumptions on the behaviour of the algorithm. In numerical experiments, it can be estimated by counting repeated runs how many times the algorithm converges. We will give some examples of such analysis. In Section 2.4 we return to the topic of efficiency and effectiveness considered simultaneously.

1.2 Efficiency

Globally efficiency is defined as the effort the algorithm needs to be successful. A usual indicator for algorithms is the (expected) number of **function evaluations** necessary to reach the optimum. This indicator depends on many factors such as the shape of the test function and the termination criteria used. The indicator more or less suggests that the calculation of function evaluations dominates the other computations of the algorithm. Several other indicators appear in literature.

In nonlinear programming (e.g. Scales (1985) and Gill et al. (1981)) the concept of **convergence speed** is common. It deals with the convergence limit of the series x_k. Let $x_0, x_1, \ldots, x_k, \ldots$ converge to point x^*. The largest number α for which

$$\lim_{k\to\infty} \frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|^\alpha} = \beta < \infty \tag{3}$$

gives the order of convergence, whereas β is called the convergence factor. In this terminology, the special instances are

- linear convergence with $\alpha = 1$ and $\beta < 1$
- quadratic convergence with $\alpha = 2$ and $0 < \beta < 1$
- superlinear convergence: $1 < \alpha < 2$ and $\beta < 1$, i.e. $\beta = 0$ when using $\alpha = 1$ in (3).

Mainly in deterministic GO algorithms, information on the past evaluations is stored in the computer memory. This requires efficient data handling for looking up necessary information during the iterations. As well **memory requirements** become a part of the computational burden as retrieving actions cannot be neglected compared to the computational effort due to function evaluations.

In stochastic GO algorithms, an efficiency indicator is the **success rate** defined as the probability that the next iterate is an improvement on the record value found thus far, i.e. $P(f(x_k) < min_{l=1,\ldots,k-1} f(x_l))$. Its theoretical relevance to convergence speed was analysed by Zabinsky and Smith (1992) and Baritompa et al. (1995), who showed that a fixed success rate of an effective algorithm (in the sense of so-called uniform covering, see e.g. Hendrix and Klepper (2000)) gives an algorithm with the expected number of function evaluations growing polynomially with the dimension of the problem. However, empirical measurements can only be established in the limit when such an algorithm stabilises, and only for specifically designed test cases Hendrix et al. (2001).

We do not go deeper into theoretical aspects here of performance indicators. Instead some basic algorithms are introduced and analysed. In Section 3, systematic investigation of algorithms is expanded upon.

2 Some basic algorithms and their goodness

2.1 Introduction

In this section several cases of algorithms are analysed for effectiveness and efficiency. Two testcases are introduced first for which the algorithms are inspected.

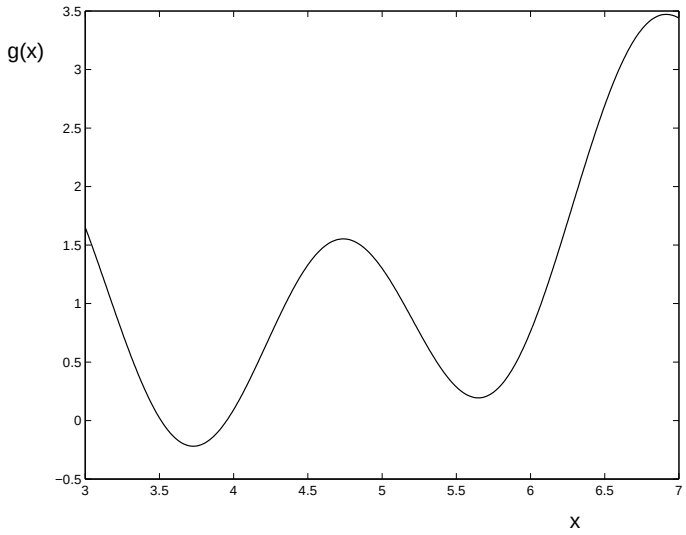

Figure 1: Test case $g(x)$ with two optima

We consider the minimisation of the following functions.

$$g(x) = \sin(x) + \sin(3x) + \ln(x), x \in [3, 7] \tag{4}$$

Function g is depicted in Figure 1 and has three minimum points on the interval. The global minimum is attained at about $x^* = 3.73$, where $f(x^*) = -0.220$. The derivative function is

$$g'(x) = \cos(x) + 3\cos(3x) + \frac{1}{x} \tag{5}$$

on the interval $[3, 7]$. Alternatively to function g, we introduce a function h with more local minimum points by adding to function g a bubble function based on $\text{frac}(x) = x - \text{round}(x)$ where $\text{round}(x)$ rounds x to the nearest integer. Now the second case is defined as

$$h(x) = g(x) + 1.5\text{frac}^2(4x) \tag{6}$$

In Figure 2, the graph of function h is shown. It has 17 local minimum points on the interval $[3, 7]$. Although g nor h are convex on the interval, at least function h is piecewise convex on the intervals in between the points of $\mathbb{S} =$

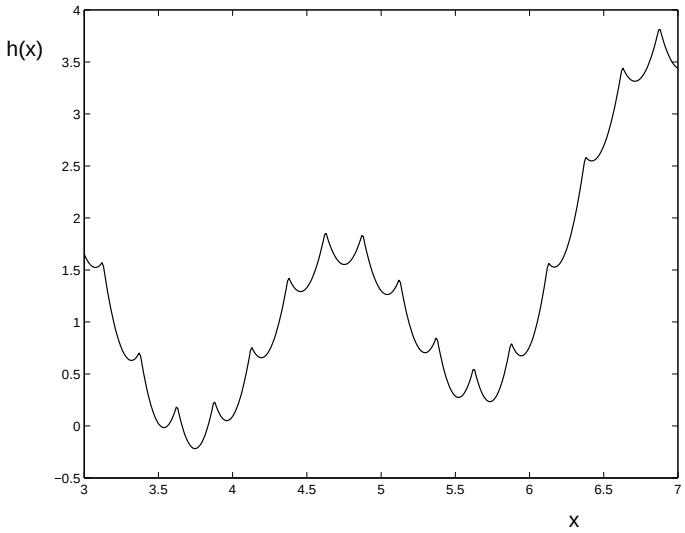

Figure 2: Test case $h(x)$ with 17 optima

$\{x = \frac{1}{4}k + \frac{1}{8}, k \in \mathbb{Z}\}$. At these points, h is not differentiable. For the rest of the interval one can define the derivative

$$h'(x) = g'(x) + 12 \times \text{frac}(4x) \quad \text{for} \quad x \notin \mathbb{S} \tag{7}$$

The global minimum point of h on $[3, 7]$ is shifted slightly compared to g towards $x^* = 3.75$, where $f(x^*) = -0.217$.

In the following Sections, we will test algorithms on their ability to find minima of these two functions. One should set a target on what is considered an acceptable or successful result. For instance one can aim at detecting a local minimum or detecting the global minimum. For the neighbourhood we will take an acceptance of $\epsilon = 0.01$. For determining an acceptable low value of the objective function we take $\delta = 0.01$. Notice that δ represents about 0.25% of the function values range.

2.2 NLP: Bisection and Newton

Two nonlinear programming algorithms are sketched and their performance measured for the two test cases. First the bisection algorithm is considered.

The algorithm departs from a starting interval $[l, r]$ that is halved iteratively based on the sign of the derivative in the midpoint. This means that the method is only applicable when the derivative is available at the generated midpoints. The point x_k converges to a minimum point within the interval $[l, r]$. If the interval contains only one minimum point, it converges to that. In our test cases, several minima exist and one can observe the convergence to one of them. The algorithm is effective in the sense of converging to a local (nonglobal)

Algorithm 1 Bisect$([l, r], f, \epsilon)$

Set $k = 0$, $l_0 = l$ and $r_0 = r$
while $(r_k - l_k > \epsilon)$
    $x_k = \frac{l_k + r_k}{2}$
    if $f'(x_k) < 0$
        $l_{k+1} = x_k$ and $r_{k+1} = r_k$
    else
        $l_{k+1} = l_k$ and $r_{k+1} = x_k$
    $k = k + 1$
End **while**

Table 1: Bisection for functions h and g, $\epsilon = 0.015$

	function h					function g				
k	l_k	r_k	x_k	$h'(x_k)$	$h(x_k)$	l_k	r_k	x_k	$g'(x_k)$	$g(x_k)$
0	3.00	7.00	5.00	-1.80	1.30	3.00	7.00	5.00	-1.80	1.30
1	5.00	7.00	6.00	3.11	0.76	5.00	7.00	6.00	3.11	0.76
2	5.00	6.00	5.50	-1.22	0.29	5.00	6.00	5.50	-1.22	0.29
3	5.50	6.00	5.75	0.95	0.24	5.50	6.00	5.75	0.95	0.24
4	5.50	5.75	5.63	-6.21	0.57	5.50	5.75	5.63	-0.21	0.20
5	5.63	5.75	5.69	-2.64	0.29	5.63	5.75	5.69	0.36	0.20
6	5.69	5.75	5.72	-0.85	0.24	5.63	5.69	5.66	0.07	0.19
7	5.72	5.75	5.73	0.05	0.23	5.63	5.66	5.64	-0.07	0.19
8	5.72	5.73	5.73	-0.40	0.23	5.64	5.66	5.65	0.00	0.19

minimum point for both cases. Another starting interval could have lead to another minimum point. In the end, we are certain that the current iterate x_k is not further away than ϵ from a minimum point. Many other stopping criteria like convergence of function values or derivatives going to zero could be used. The current stopping criterion is easy for analysis of efficiency. One question could be: How many iterations (i.e. corresponding (derivative) function evaluations) are necessary to come closer than ϵ to a minimum point. The bisection algorithm is a typical case of linear convergence with a convergence factor of $\frac{1}{2}$, $\frac{|r_{k+1} - l_{k+1}|}{|r_k - l_k|} = \frac{1}{2}$. This means one can determine the number of iterations necessary for reaching ϵ-convergence:

$$
\begin{aligned}
&| r_k - l_k | = (\tfrac{1}{2})^k \times | r_0 - l_0 | < \epsilon \\
&(\tfrac{1}{2})^k < \tfrac{\epsilon}{|r_0 - l_0|} \\
&k > \tfrac{\ln \epsilon - \ln |r_0 - l_0|}{ln \frac{1}{2}}
\end{aligned}
$$

The example case requires at least 12 iterations to reach an accuracy of $\epsilon = 0.01$.

An alternative for finding the zero point of an equation, in our case the derivative, is the so-called method of **Newton**. The idea is that its efficiency

is known to be superlinear (e.g. Scales (1985)), so it should be faster than bisection. We analyse its efficiency and effectiveness for the two test cases. In

Algorithm 2 Newt$([l, r], x_0, f, \alpha)$

Set $k = 0$,
while $(\mid f'(x_k) \mid > \alpha)$
$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$
! safeguard for staying in interval
if $x_{k+1} < l$, $x_{k+1} = l$
if $x_{k+1} > r$, $x_{k+1} = r$
if $x_{k+1} = x_k$, STOP
$k = k + 1$
End **while**

general, the aim of the Newton algorithm is to converge to a point where the derivative is zero. Depending on the starting point x_0, the method may converge to a minimum or maximum. Also, it may not converge at all, for instance when a minimum point does not exist. Specifically in the version of Algorithm 2, a safeguard is built in to ensure the iterates remain in the interval; it can converge to a boundary point. If x_0 is in the neighbourhood of a minimum point where f is convex, then convergence is guaranteed and the algorithm is effective in the sense of reaching a minimum point. Let us consider what happens for the two test cases. When choosing the starting point x_0 in the middle of the interval

Table 2: Newton for functions h and g, $\alpha = 0.001$

	function h				function g			
k	x_k	$h'(x_k)$	$h''(x_k)$	$h(x_k)$	x_k	$g'(x_k)$	$g''(x_k)$	$g(x_k)$
0	5.000	-1.795	43.066	1.301	5.000	-1.795	-4.934	1.301
1	5.042	0.018	43.953	1.264	4.636	0.820	-7.815	1.511
2	5.041	0.000	43.944	1.264	4.741	-0.018	-8.012	1.553
3	5.041	0.000	43.944	1.264	4.739	0.000	-8.017	1.553

$[3, 7]$, the algorithm converges to the closest minimum point for function h and to a maximum point for the function g, i.e. it fails for this starting point. This gives rise to introducing the concept of a **region of attraction** of a minimum point x^*. A region of attraction of point x^*, is the region of starting points x_0 where the local search procedure converges to point x^*. We elaborate this concept further in Section 2.4.

One can observe here when experimenting further, that when x_0 is close to a minimum point of g, the algorithm converges to one of the minimum points. Morever, notice now the effect of the safeguard to keep the iterates in the interval $[3, 7]$. If for instance $x_{k+1} < 3$, it is forced to a value of 3. In this way, also the

left point $l = 3$ is an attraction point of the algorithm. Function h is piecewise convex, such that the algorithm always converges to the closest minimum point.

2.3 Deterministic GO: Grid search, Piyavskii-Shubert

The aim of deterministic GO algorithms is to approach the optimum with a given certainty. We sketch two algorithms for the analysis of effectiveness and efficiency. The idea of reaching the optimum with an accuracy of ϵ can be done by so-called "everywhere dense sampling", as introduced in literature on Global Optimisation (see e.g.Törn and Zilinskas (1989)). In a rectangular domain this can be done by constructing a grid with a mesh of ϵ. By evaluating all points on the grid, the best point found is a nice approximation of the global minimum point. The difficulty of GO is, that even this best point found may be far away from the global minimum point, as the function may have a needle shape in another region in between the grid points. As shown in literature, one can always construct a polynomial of sufficiently high degree, that fits all the evaluated points and has a minimum point more than ϵ away from the best point found. Actually, grid search is theoretically not effective if no further assumptions are posed on the optimisation problem to be solved.

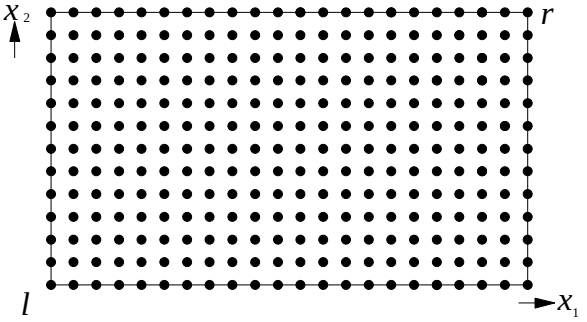

Figure 3: Equidistant grid over rectangular feasible set

Let us have a look at the behaviour of the algorithm for our two cases. For the ease of the formulation we write down the grid algorithm for one dimensional functions. The algorithm starts with the domain $[l, r]$ written as an interval and generates $M = \lceil (r-l)/\epsilon \rceil + 1$ grid points, where $\lceil x \rceil$ is the lowest integer greater than or equal to x. Experimenting with test functions g and h gives reasonable

Algorithm 3 Grid$([l, r], f, \epsilon)$

$M = \lceil (r-l)/\epsilon \rceil + 1$, $f^U = \infty$
for $(k = 1$ to $M)$ do
    $x_k = l + \frac{(k-1)\times(r-l)}{M-1}$
    if $f(x_k) < f^U$
        $f^U = f(x_k)$ and $x^U = x_k$
End **for**

results for $\epsilon = 0.01$, $(M = 401)$ and $\epsilon = 0.1$, $(M = 41)$. In both cases one finds an approximation x^U less than ϵ from the global minimum point. One knows exactly how many function evaluations are required to reach this result in advance.

The efficiency of the algorithm in higher dimensions is also easy to establish. Given the lower left vector l and upper right vector r of a rectangular domain, one can determine easily how many grid co-ordinates $M_j, j = 1, \ldots, n$ in each direction should be taken and the total number of grid points is $\prod_j M_j$. This number is growing exponentially in the dimension n. As mentioned before, the effectiveness is not guaranteed in the sense of being closer than ϵ from a global minimum point, unless we make an assumption on the behaviour of the function. A usual assumption in the literature is Lipschitz continuity.

Definition 2.1 *L is called a Lipschitz constant of f on X if:*

$$| f(x) - f(y) | \leq L\|x - y\|, \quad \forall x, y \in X$$

In a practical sense it means that big jumps do not appear in the function value; slopes are bounded. With such an assumption, the δ-accuracy in the function space translates into an ϵ-accuracy in the x-space. Choosing $\epsilon = \delta/L$ gives that the best point x^U is in function value finally close to minimum point x^*:

$$| f^U - f^* | \leq L\|x^U - x^*\| \leq L\epsilon = \delta \tag{8}$$

In higher dimension, one should be more exact in the choice of the distance

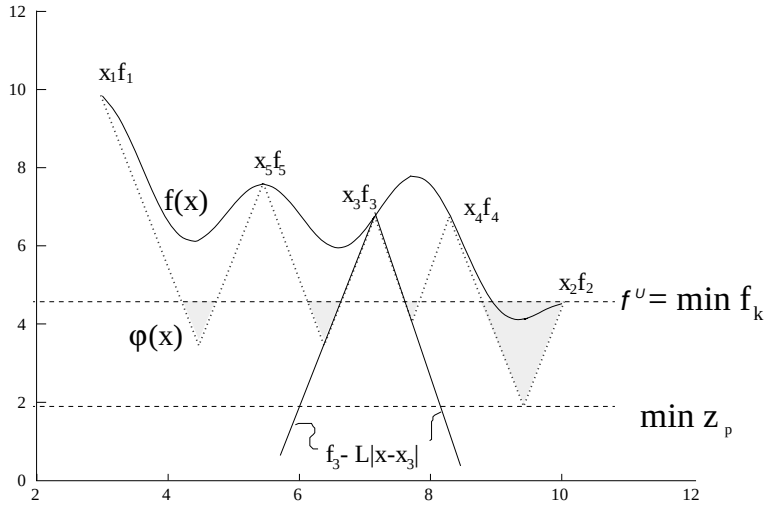

Figure 4: Piyavskii-Shubert algorithm

norm $\|\cdot\|$. Here, for the one-dimensional examples we can focus on deriving

the accuracy for our cases in a simple way. For a one-dimensional differentiable function f, L can be taken as

$$L = \max_{x \in X} \mid f'(x) \mid \tag{9}$$

Using equation (9), one can now derive valid estimates for the example functions h and g. One can derive an over estimate L_g for the Lipschitz constant of g on $[3, 7]$ as

$$\begin{array}{l} \max_{x\in[3,7]} \mid g'(x) \mid = \max_{x\in[3,7]} \mid \cos(x) + 3\cos(3x) + \frac{1}{x} \mid \ \leq \\ \max_{x\in[3,7]} \{ \mid \cos(x) \mid + \mid 3\cos(3x) \mid + \mid \frac{1}{x} \mid \} \leq \\ \max_{x\in[3,7]} \mid \cos(x) \mid + \max_{x\in[3,7]} \mid 3\cos(3x) \mid + \max_{x\in[3,7]} \mid \frac{1}{x} \mid = \\ 1 + 3 + \frac{1}{3} = L_g \end{array} \tag{10}$$

The estimate of L_h based on (7) is done by adding the maximum derivative of the bubble function $12 \times \frac{1}{2}$ to L_g for illustrative purposes rounded down to $L_h = 10$. We can now use (8) to derive a guarantee for the accuracy. One arrives certainly closer than $\delta = 0.01$ to the minimum in function value by taking for function g a mesh size of $\epsilon = \frac{0.01}{4.33} = 0.0023$ and for function h taking $\epsilon = 0.001$. For the efficiency of grid search this means that reaching the δ-guarantee requires the evaluation of $M = 1733$ points for function g and $M = 4001$ points for function h. Note, that due to the one dimensional nature of the cases, ϵ can be taken twice as big, as the optimum point x^* is closer than half the mesh size to an evaluated point.

The main idea of deterministic algorithms is not to generate and evaluate points everywhere dense, but to throw out those regions, where the optimum cannot be situated. Giving a Lipschitz constant, independently **Piyavskii and Shubert** constructed a similar algorithm, see Shubert (1972) and Danilin and Piyavskii (1967). From the point of view of the graph of the function f to be minimised and an evaluated point (x_k, f_k), one can say that the region described by $x, f < f_k - L \mid x - x_k \mid$ cannot contain the optimum; the graph is above the function $f_k - L \mid x - x_k \mid$. Given a set of evaluated points $\{x_k\}$, one can construct a lower bounding function, a so-called saw-tooth underestimator that is given by $\varphi(x) = \max_k(f_k - L \mid x - x_k \mid)$ as illustrated by figure 4. Given that we have also an upper bound f^U on the minimum of f being the best function value found thus far, one can say that the minimum point has to be in one of the shaded areas. We will describe here the algorithm from a Branch-and-Bound point of view, where the subsets are defined by intervals $[l_p, r_p]$ and the end points are given by evaluated points. The index p is used to represent the intervals in Λ. For each interval, a lower bound is given by

$$z_p = \frac{f(l_p) + f(r_p)}{2} - \frac{L(r_p - l_p)}{2} \tag{11}$$

The gain with respect to grid search is that an interval can be thrown out as soon as $z_p > f^U$. Moreover, δ works as a stopping criterion as the algorithm

Algorithm 4 PiyavShub($[l, r], f, L, \delta$)

Set $p = 1$, $l_1 = l$ and $r_1 = r, \Lambda = \{[l_1, r_1]\}$
$z_1 = \frac{f(l_1)+f(r_1)}{2} - \frac{L(r_1-l_1)}{2}, f^U = \min\{f(l), f(r)\}, x^U = argmin\{f(l), f(r)\}$
while $(\Lambda \neq \emptyset)$
    remove an interval $[l_k, r_k]$ from Λ with $z_k = \min_p z_p$
    evaluate $f(m_k) = f(\frac{f(l_k)-f(r_k)}{2L} + \frac{r_k+l_k}{2})$
    if $f(m_k) < f^U$
        $f^U = f(m_k), x^U = m_k$ and remove all C_p from Λ with $z_p > f^U - \delta$
    split $[l_k, r_k]$ into 2 new intervals $C_{p+1} = [l_k, m_k]$ and$C_{p+2} = [m_k, r_k]$
        with corresponding lower bounds z_{p+1} and z_{p+2}
    if $z_{p+1} < f^U - \delta$ store C_{p+1} in Λ
    if $z_{p+2} < f^U - \delta$ store C_{p+2} in Λ
    $p = p + 2$
End **while**

implicitly (by not storing) compares the gap between f^U and $\min_p z_p$; stop if $(f^U - \min_p z_p) < \delta$. The algorithm proceeds by selecting the interval corresponding to $\min_p z_p$ (most promising) and splitting it over the minimum point of the saw tooth cover $\varphi(x)$ defined by:

$$m_p = \frac{f(l_p) - f(r_p)}{2L} + \frac{r_p + l_p}{2} \tag{12}$$

being the next point to be evaluated. By continuing evaluating, splitting and throwing out intervals where the optimum cannot be, finally the stopping criterion is reached and we are certain to be closer than δ from f^* and therefore closer than $\epsilon = \delta/L$ from one of the global minimum points. The consequence of using such an algorithm, in contrast to the other algorithms, is that we now have to store information in a computer consisting of a list Λ of intervals. This computational effort is now added to that of evaluating sample points and doing intermediate calculations. This concept becomes more clear when running the algorithm on the test function g using an accuracy of $\delta = 0.01$. The Lipschitz constant $L_g = 4.2$ is used for illustrative purposes. As can be read from Table 3, the algorithm is slowly converging. After some iterations, 15 intervals have been generated of which 6 are stored and 2 can be discarded due to the bounding; it has been proven that the minimum cannot be in the interval $[5.67, 7]$. The current estimate of the optimum is $x^U = 3.66, f^U = -0.19$ and the current lower bound is given by $\min_p z_p = -0.68$. Figure 5 illustrates the appearing binary structure of the search tree.

The maximum computational effort with respect to storing intervals is reached when the branching proceeds and no parts can be thrown out; 2^K intervals appear at the bottom of the tree, where K is the depth of the tree. This mainly happens when the used Lipschitz parameter L is overestimating the maximum

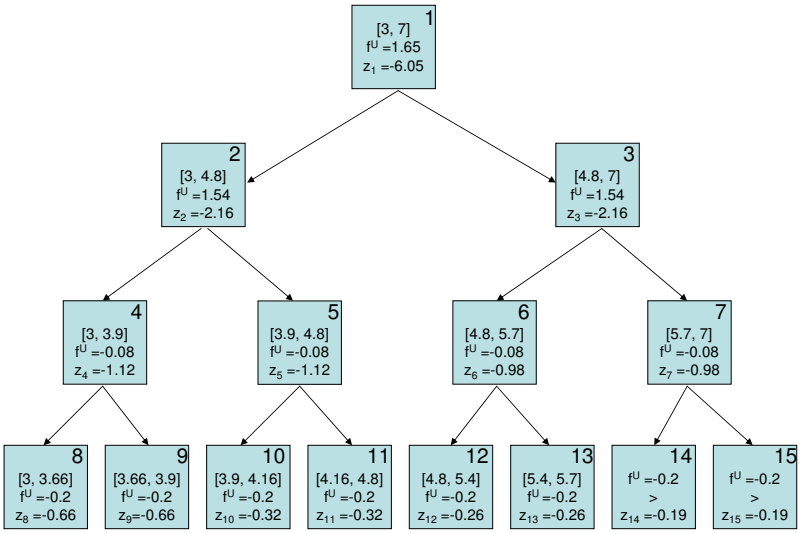

Figure 5: Branch-and-Bound tree of Piyavskii-Shubert for function g

slope drastically, or seen in the other way, the function is very flat compared to the used constant L. In that case, more than the M points of the regular grid are evaluated. With a correct constant L, the number of evaluated points is less as part of the domain can be discarded as illustrated here.

2.4 Stochastic GO: PRS, Multistart, Simulated Annealing

Stochastic methods are understood to contain some stochastic elements. Either the outcome of the method is a random variable or the objective function itself is considered a realisation of a stochastic process. For an overview on stochastic methods we refer to Törn and Zilinskas (1989) and Boender and Romeijn (1995). Two classical approaches from Global Optimization, Pure Random Search (PRS) and Multistart are analysed for the test cases. This is followed by a classical variant of Simulated Annealing, a so-called heuristic.

Pure Random Search (PRS) generates points uniformly over the domain and stores the point corresponding to the best value as the approximation of the global minimum point. The algorithm is popular as a reference algorithm as it can easily be analysed. The question can now be how it behaves for our test cases g and h. The domain is clearly the interval $[3, 7]$, but what can be defined as the success region now? If the success is defined as the idea that one of the generated points is closer than $\epsilon = 0.01$ to the global minimum point, the probablity we did NOT hit this region after $N = 50$ trials is $(3.98/4)^{50} \approx 0.78$. In the specific case, the size of the success region is namely $2 \times \epsilon$ and the size

Table 3: Piyavskii-Shubert for function g, $\delta = 0.01$

p	l_p	r_p	$f(l_p)$	$f(r_p)$	m_p	z_p	f^U	x^U	
1	3.00	7.00	1.65	3.44	4.79	-5.85	1.65	3.00	split
2	3.00	4.79	1.65	1.54	3.91	-2.16	1.54	4.79	split
3	4.79	7.00	1.54	3.44	5.67	-2.16	1.54	4.79	split
4	3.00	3.91	1.65	-0.08	3.66	-1.12	-0.08	3.91	split
5	3.91	4.79	-0.08	1.54	4.15	-1.12	-0.08	3.91	split
6	4.79	5.67	1.54	0.20	5.39	-0.98	-0.08	3.91	split
7	5.67	7.00	0.20	3.44	5.95	-0.98	-0.08	3.91	split
8	3.00	3.66	1.65	-0.20	3.55	-0.66	-0.20	3.66	
9	3.66	3.91	-0.20	-0.08	3.77	-0.66	-0.20	3.66	
10	3.91	4.15	-0.08	0.47	3.96	-0.32	-0.20	3.66	
11	4.15	4.79	0.47	1.54	4.34	-0.32	-0.20	3.66	
12	4.79	5.39	1.54	0.46	5.22	-0.26	-0.20	3.66	
13	5.39	5.67	0.46	0.20	5.56	-0.26	-0.20	3.66	
14	5.67	5.95	0.20	0.61	5.76	-0.19	-0.20	3.66	discarded
15	5.95	7.00	0.61	3.44	6.14	-0.19	-0.20	3.66	discarded

Algorithm 5 PRS(X, f, N)

$f^U = \infty$
for $(k = 1$ to $N)$ do
  Generate x_k uniformly over X
  if $f(x_k) < f^U$
    $f^U = f(x_k)$ and $x^U = x_k$
End **for**

of the feasible area is 4. The probability of NOT hitting $(1 - \frac{0.02}{4})$ and of NOT hitting 50 times is $(1 - \frac{0.02}{4})^{50}$. This means that the probability of success as efficiency indicator has a value of about 0.22 for both cases h and g.

A similar analysis can be done for determining the probability that the function value of PRS after $N = 50$ iterations is less than $f^* + \delta$ for $\delta = 0.01$. The usual tool in the analysis on the function space is to introduce $\boldsymbol{y}$=$f(\boldsymbol{x})$ as a random variate representing the function value, where $\boldsymbol{x}$ is uniformly distributed over X. Value $\boldsymbol{y}$ has distribution function $F(y) = P(f(\boldsymbol{x}) \leq y)$. Keeping this in mind, analysis with so-called extreme order statistics has shown that the outcome of PRS as record value of N points can be easily derived from $F(y)$. For a complete introduction into extreme order statistics in optimisation, we refer to Zhigljavsky (1991). Under mild assumptions it can be shown that $\boldsymbol{y}_{(1)} = \min\{f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_N)\}$ has the distribution function $F_{(1)}(y) = 1 - (1 - F(y))^N$. This means that for the question about the probability that $\boldsymbol{y}_{(1)} \leq f^* + \delta$, we dont have to know the complete distribution function F, but only the probability mass $F(f^* + \delta)$ of the success level set where $f(x) \leq f^* + \delta$, i.e. the

probability that one sample point hits this low level set. Here the 2 test cases differ considerably. One can can verify that the level set of the more smooth test function g is about 0.09 wide, whereas that of function h is only 0.04 wide for a δ of 0.01. This means that the probability of PRS to reach a level below $f^* + \delta$ after 50 evaluations for function g is $1 - (1 - \frac{0.09}{4})^{50} = 0.68$, whereas the same probability for function h is $1 - (1 - \frac{0.04}{4})^{50} = 0.40$.

An early obvservation based on the extreme order statistic analysis is due to Karnopp (1963). Surprisingly enough, Karnopp showed that the probability of finding a better function value with one draw more after N points have been generated, is $\frac{1}{N+1}$, independent of the problem to be solved. Generating K more points increases the probability to $\frac{K}{N+K}$. The derivation which also can be found in Törn and Zilinskas (1989) is based on extreme order statistics and only requires F not to behave too strange, e.g. F is continuous such that f should not have plateaus on the domain X.

In conclusion, stochastic algorithms show something which in the literature is called the **infinite effort property**. This means that if one proceeds long enough (read $N \to \infty$) in the end the global optimum is found. The problem with such a concept is that infinity can be pretty far away. Moreover, we have seen in the earlier analyses that the probability of reaching what one wants, depends considerably on the size of the success region. One classical way of increasing the probability of reaching an optimum is to use (nonlinear optimisation) local searches. This method is called **multistart**.

Define a local optimisation routine $LS(x) : X \to X$ as a procedure which given a starting point returns a point in the domain that approximates a local minimum point. As an example, one can consider the Newton method of Section 2.2. Multistart generates convergence points of a local optimisation routine from randomly generated starting points.

Algorithm 6 Multistart(X, f, LS, N)

$f^U = \infty$
for $(k = 1$ to $N)$ do
    Generate x uniformly over X
    $x_k = LS(x)$
    if $f(x_k) < f^U$
        $f^U = f(x_k)$ and $x^U = x_k$
End **for**

Notice that the number of iterations N is not comparable with that in PRS, as every local search requires several function evaluations. Let us for the example cases assume that the Newton algorithm requires 5 function evaluations to detect an attraction point, as is also implied by table 2. As we were using $N = 50$ function evaluations to assess the success of PRS on the test cases, we will use $N = 10$ iterations for Multistart. In order to determine a similar probability of

success, one should find the relative size of the region of attraction of the global minimum point.

For function g, the region of attraction is not easy to determine. It consists of a range of about 0.8 on the feasible area of size 4, such that the probability one random starting point leads to success is $0.8/4 = 0.2$. For function h, the good region of attraction is simply the bubble of size 0.25 around the global minimum point, such that the probability of finding the global minimum in one iteration is about 0.06. Reaching the optimum after $N = 10$ restarts is $1 - 0.8^{10} \approx 0.89$ for g and $1 - 0.94^{10} \approx 0.48$ for h. In both examples, the probability of success is larger than that of PRS.

As sketched sofar, the algorithms of Pure Random Search and Multistart have been analysed widely in the literature of GO. Algorithms that are far less easy to analyse, but very popular in applications are the collection of so-called metaheuristics. This term was introduced by Fred Glover in Glover (1986) and includes Simulated Annealing, Evolutionary algorithms, Genetic algorithms, tabu search and all fantasy names derived from crossovers of the other names. Originally such algorithms were not only aimed at continuous optimisation problems, see Aarts and Lenstra (1997). An interesting research question is whether they are really better than combining classical ideas of random search and nonlinear optimisation local searches. We discuss here a variant of **Simulated annealing**, a concept that also got attention in the GO literature, see Romeijn (1992). Simulated annealing describes a sampling process in the decision space where new sample points are generated from a so-called neighbourhood of the current iterate. The new sample point is always accepted when it is better and with a certain probability when it is worse. The probability depends on the so-called temperature that is decreasing (cooling) during the iterations. The

Algorithm 7 SA(X, f, CR, N)

$f^U = \infty$, $T_1 = 1000$
Generate x_1 uniformly over X
for $(k = 1$ to $N)$ do
    Generate x from a neighbourhood of x_k
    if $f(x) < f(x_k)$
        $x_{k+1} = x$
        if $f(x) < f^U$
            $f^U = f(x)$ and $x^U = x$
    else with probability $e^{\frac{f(x_k)-f(x)}{T_k}}$ let $x_{k+1} = x$
    $T_{k+1} = CR \times T_k$
End **for**

algorithm contains the parameter CR representing the *Cooling rate* with which the temperature variable decreases. A fixed value of 1000 was taken for the initial temperature to avoid creating another algorithm parameter. The algorithm

accepts a worse point depending on how much it is worse and the development of the algorithm. This is a generic concept in Simulated Annealing. There are several ways to implement the concept of "sample from neighbourhood". In one dimension one would perceive intuitively a neighbourhood of x_k in real space $[x_k - \epsilon, x_k + \epsilon]$, which can be found in many algorithms e.g. see Baritompa et al. (2005). As originally such heuristics were not aiming at continuous optimisation problems, but at integer problems, one of the first approaches was the coding of continuous variables in bit strings. For the illustrations, we elaborate this idea for the test case. Each point $x \in [3, 7]$ is represented by a bitstring $(B_1, \ldots, B_9) \in \{0, 1\}^9$ where,

$$x = 3 + 4\frac{\sum_{i=1}^{9} B_i 2^{i-1}}{511} \tag{13}$$

Formula (13) describes a regular grid over the interval, where each of the M=512 bitstrings is one of the grid points, such that the mesh size is $\frac{4}{511}$. The sampling from a neighbourhood of a point x is done by flipping at random one of its bit variables B_i from a value of 0 to 1, or the other way around. Notice that by doing so, the generated point is not necessarily in what one would perceive as a neighbourhood in continuous space. The question is therefore, whether the described SA variant will perform better than an algorithm where the new sample point does not depend on the current iterate, PRS. To test this, a figure is introduced that is quite common in experimenting with meta-heuristics. It is a graph with on the x-axis the effort and on the y-axis the reached success. The GO literature often looks at the two criteria effectiveness and efficiency separately. Figure 6 being a special case of what in Baritompa and Hendrix (2005) was called the **performance graph**, gives a trade-off between the two main criteria. One can also consider the x-axis to give a budget with which one has to reach a level as low as possible, see Hendrix and Roosma (1996). In this way one can change the search strategy depending on the amount of available running time. The figure suggests for instance that a high cooling rate (the process looks like PRS) is doing better for a lower number of function values and doing worse for a higher number of function values. Figure 6 gives an estimation of the expected level one can reach by running SA on function g. Implicitly it says the user wants to reach a low function value; not necessarily a global minimum point. Theoretically, one can derive the expected level analytically by considering the process from a Markov chain perspective, see e.g. Bulger and Wood (1998). However, usually the estimation is done empirically and the figure is therefore very common in metaheuristic approaches. The reader will not be surprised that the figure looks similar for function h, as the number of local optima is not relevant for the bitstring perspective and the function value distribution is similar. Theoretically, one can also derive the expected value of the minimum function value reached by PRS. It is easier to consider the theoretical behaviour from the perspective where success is defined boolean, as has been done so far.

Let us consider again the situation that the algorithm reaches the global optimum as success. For stochastic algorithms we are interested in the probability

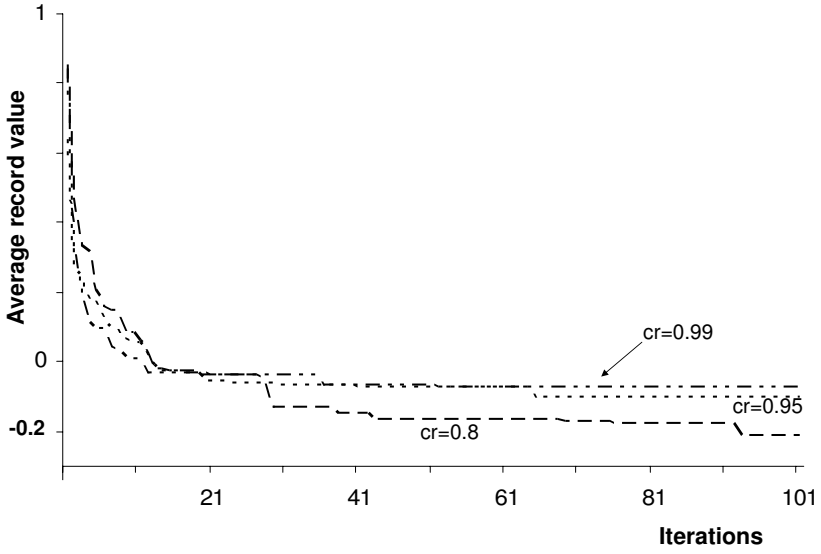

Figure 6: Average record value over 10 runs of SA, 3 different values of CR for a given amount of function evaluations, test case g

of success. Define reaching the optimum again as finding a point with function value closer than $\delta = 0.1$ to the minimum. For function g this is about 2.2% (11 out of the 512 bitstrings) of the domain. For PRS, one can determine the probability of success as $P_{PRS}(N) = 1 - 0.978^N$. For SA this is much harder to determine, but one can estimate the probability of success empirically. The result is the performance graph in Figure 7. Let us have a look at the figure critically. In fact it suggests that PRS is doing as good as the SA algorithms. As this is verifying a hypothesis (not falsifying), this is a reason to be suspicious. The following critical remarks can be made.

- the 10 runs are enough to illustrate how the performance can be estimated, but is too low to discriminate between methods. Perhaps the author has even selected a set of runs which fits the hypothesis nicely.
- one can choose the scale of the axes to focus on an effect. In this case, one can observe that up to 40 iterations, PRS does not look better than the SA variants. By choosing the x-axis to run to 100 iterations, it looks much better.
- the graph has been depicted for function g, but not for function h, where the size of the success region is twice as small. One can verify, that in the given range, the SA variants are nearly always doing better.

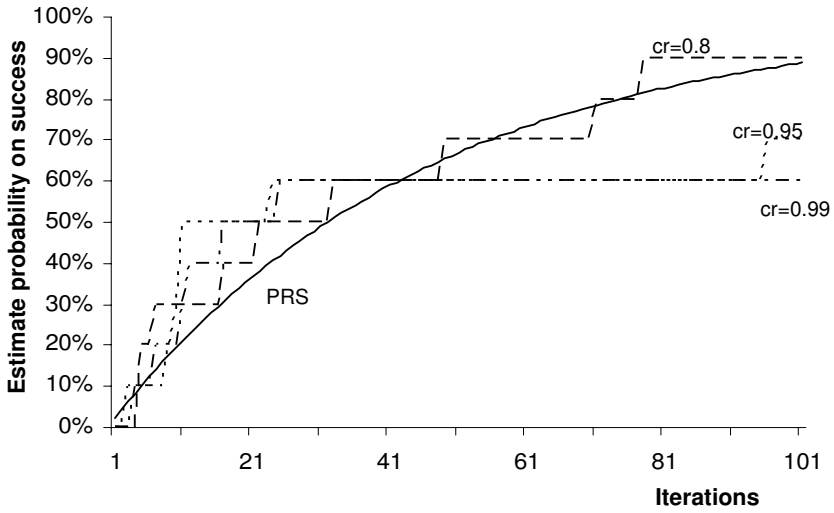

Figure 7: Estimate of probability of reaching the minimum for PRS and SA (average over 10 runs) on function g given a number of function evaluations

This brings us to the general scientific remark, that all results should be described in such a way that they can be **reproduced** i.e. repeating the exercise will yield similar results. For the exercises reported in this section, this is relatively easy. Spreadsheet calculations and for instance matlab implementations can easily be made.

3 Investigating algorithms

In Section 2, we have seen how several algorithms behave on some test cases. What did we learn from that? How to do the investigation systematically? Figure 8 depicts some relevant aspects. All aspects should be considered together. Baritompa and Hendrix (2005) distinguish the following steps.

1. Formulation of performance criteria.
2. Description of the algorithm(s) under investigation.
3. Selection of appropriate algorithm parameters.
4. Production of test functions (instances, special cases) corresponding to certain landscape **structures or characteristics**.
5. Analysis of its theoretical performance, or empirical testing.

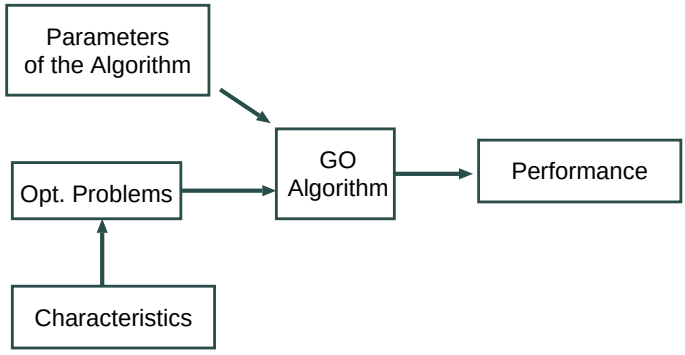

Figure 8: Aspects of investigating Global Optimization Algorithms

Several criteria and performance indicators have been sketched: to get a low value, to reach a local minimum point, a high probability to hit an ϵ neighborhood of a global minimum point, obtain a guarantee to be less than δ from the minimum function value, etc. Several types of algorithms have been outlined. The number of parameters has been kept low, a Lipschitz constant L, the number of iterations N, cooling rate CR, stopping accuracy α. Mainly modern heuristics contain so many tuning parameters, that it is hard to determine the effect of their value on the performance of the algorithm.

Only two test functions were introduced to experiment with. The main difference between them is the number of optima, which in literature is seen as an important characteristic. However, in the illustrations, the number of optima was only important for the performance of multistart. The piecewise convexity appeared important for the local search Newton algorithm and further mainly the difference in size of the δ-level set was of importance for the probability of success of stochastic algorithms. It teaches us, that in a research setting, one should think carefully, make hypotheses and design corresponding experiments, to determine which characteristics of test functions are relevant for the algorithm under investigation.

3.1 Characteristics

In Baritompa et al. (2005), an attempt is made to analyse the interrelation between the landscape describing the characteristics of the test cases and the behaviour of the algorithms. What we see experimentally, is that often an algorithm is run over several test functions and its performance compared to other algorithms and/or other parameter settings. To understand behaviour, we need to study the relationships to characteristics of landscapes of test functions. The main question is how to define appropriate characteristics. We will discuss some ideas which appear in literature. The main idea, as illustrated before, is that relevant characteristics depend on the type of algorithm as well as on the performance measure. Extending previous stochastic examples to more

dimensions, it is only the relative size of the sought for region that matters, characteristics such as the number of optima, the shape of regions of attraction, the form of the level sets, barriers in the landscape do not matter.

It is also important to vary the test cases systematically between the extreme cases, in order to understand how algorithms behave. In an experimental setting, depending on what one measures, one tries to design experiments which yield as much information as possible. To derive analytical results, it is not uncommon to make highly specific assumptions which make the analysis tractable.

When studying cases, we should keep in mind that in the GO literature the following types of problems have been investigated.

- Black box (or oracle) case: in this case it is assumed that nothing is known about the function to be optimized. Often the feasible set is defined as a box, but information about the objective function can only be obtained by evaluating the function at feasible points.

- Grey box case: something is known about the function, but the explicit form is not necessarily given. We may have a lower bound on the function value or on the number of global and/or local optima. As has proven useful for deterministic methods, we may have structural information such as: a concave function, a known Lipschitz constant, a known so-called d.c-decomposition. Stochastic methods often don't require this type of information, but this information may be used to derive analytical or experimental results.

- White box case: explicit analytical expressions of the problem to be solved are assumed to be available. Specifically so-called interval arithmetic algorithms require this point of view on the problem to be solved.

When looking at the structure of the instances for which one studies the behaviour of the algorithm, we should keep two things in mind.

- In experiments, the researcher can try to influence the characteristics of the test cases such that the effect on what is measured is as big as possible. Note that the experimentalist knows the structure in advance, but the algorithm doesn't.

- The algorithm can try to generate information which tells it about the landscape of the problems. We will enumerate some information which can be measured in the black box case.

When we have a look at the lists of test functions in literature (a.o. Törn and Zilinskas (1989)), we observe as characteristics the number of Global minimum points, the number of local minimum points and the dimension of the problem.

A difficulty in the analysis of a GO algorithm in the multiextremal case is, that everything seems to influence behaviour: The orientation of components of lower level sets with respect to each other determine how iterates can jump from one place to the other. The number of local optima up in the "hills" determine

how algorithms may get stuck in local optima. The difference between the global minimum and the next lowest minimum affects the ability of detecting the global minimum point. The steepness around minimum points, valleys, creeks etc. which determine the landscape influences the success. However, we stress, as shown by the examples, the **characteristics** which are important for the behaviour depend on the **type of algorithm** and the **performance criteria** that describe the behaviour.

In general, stochastic algorithms require no structural information about the problem. However, one can adapt algorithms to make use of structure information. Moreover, one should notice, that even if structural information is not available, other so-called **value information**, becomes available when running algorithms: the number of local optima found thus far, the average number of function evaluations necessary for one local search, best function value found, and the behavior of the local search etc. Such indicators can be measured empirically and can be used to get insight into what factors determine the behaviour of a particular algorithm and perhaps can be used to improve the performance of an algorithm. From the perspective of designing algorithms, running them empirically may **generate information** about the landscape of the problem to be solved. A list of information one could measure during running a stochastic GO algorithm on a black box case from Hendrix (1998) is useful:

- Graphical information on the decision space.
- Current function value.
- Best function value found so far (record).
- Number of evaluations in the current local phase.
- Number of optima found.
- Number of times each detected minimum point is found.
- Estimates of the time of one function evaluation.
- Estimates of the number of function evaluations for one local search.
- Indicators on the likelihood to have found an optimal solution

For the latter indicator, a probability model is needed. Measuring and using the information in the algorithm, usually leads to more extended algorithms, called "adaptive". Often, they have additional parameters complicating the analysis of what are good parameter settings.

3.2 Comparison of Algorithms

When comparing algorithms, a specific algorithm is **dominated**, if there is another algorithm which performs better (e.g. has a higher probability performance graph) in all possible cases under consideration. Usually however, one algorithm runs better on some cases and another on other cases.

So basically, the performance of algorithms can be compared on the same test function, or preferably for many test functions with the same characteristic, where that characteristic is the only parameter that matters for the performance

of the compared algorithms. As we have seen, it may be very hard to find out such characteristics. The following principles can be useful:

- Comparability: When comparing several algorithms, they should all make use of the same type of (structural) information (same stopping criteria, accuracies etc).
- Simple references: It is wise to include in the comparison simple benchmark algorithms such as Pure Random Search, Multistart and Grid Search in order not to let analysis of the outcomes get lost in parameter tuning and complicated schemes.
- Reproducibility: In principle, the description of the method that is used to generate the results, has to be so complete, that someone else gets similar results (not necessarily the same) when repeating the exercise.

Often in applied literature, we see algorithms used for solving "practical" problems. If we are comparing algorithms for a practical problem, we should keep in mind this is only one problem and up to now, nobody has defined what is a representative practical problem.

4 Summary and discussion points

- Results of investigation of GO algorithms consist of a description of the performance appropriate to the algorithms (parameter settings) and characteristics of test functions or function classes.
- To obtain good performance criteria, one needs to identify the target of an assumed user and to define what is considered as success.
- The performance graph is a useful instrument to compare performance.
- The relevant characteristics depend on the type of algorithm and performance criterion under consideration.
- Algorithms to be comparable must make use of same information, accuracies, principles etc.
- It is wise to include simple benchmark algorithms like Grid Search, PRS and Multistart as references.
- One should be able to repeat reported methods and to obtain similar results.

Acknowledgement
Special thanks are due to Joke van Lemmen and Bill Baritompa for reading and commenting on earlier versions of the manuscript.

References

Aarts, E. H. L. and Lenstra, J. K.: 1997, *Local Search Algorithms*, Wiley, New York.

Baritompa, W. P., Dür, M., Hendrix, E. M. T., Noakes, L., Pullan, W. and Wood, G. R.: 2005, Matching stochastic algorithms to objective function landscapes, *Journal of Global Optimization* **31**, 579–598.

Baritompa, W. P. and Hendrix, E. M. T.: 2005, On the investigation of stochastic global optimization algorithms, *Journal of Global Optimization* **31**, 567–578.

Baritompa, W. P., Mladineo, R., Wood, G. R., Zabinsky, Z. B. and Baoping, Z.: 1995, Towards pure adaptive search, *Journal of Global Optimization* **7**, 73–110.

Boender, C. G. E. and Romeijn, H. E.: 1995, Stochastic methods, *in* R. Horst and P. M. Pardalos (eds), *Handbook of Global Optimization*, Kluwer, Dordrecht, pp. 829–871.

Bulger, D. W. and Wood, G. R.: 1998, Hesitant adaptive search for global optimisation, *Mathematical Programming* **81**, 89–102.

Danilin, Y. and Piyavskii, S. A.: 1967, An algorithm for finding the absolute minimum, *Theory of Optimal Decisions* **2**, 25–37. (in Russian).

Gill, P. E., Murray, W. and Wright, M. H.: 1981, *Practical Optimization*, Academic Press, New York.

Glover, F. W.: 1986, Future paths for integer programming and link to artificial intelligence, *Computers and Operations Research* **13**, 533–554.

Hendrix, E. M. T.: 1998, *Global Optimization at Work*, PhD thesis, Wageningen University, Wageningen.

Hendrix, E. M. T. and Klepper, O.: 2000, On uniform covering, adaptive random search and raspberries, *Journal of Global Optimization* **18**, 143–163.

Hendrix, E. M. T., Ortigosa, P. M. and García, I.: 2001, On success rates for controlled random search, *Journal of Global Optimization* **21**, 239–263.

Hendrix, E. M. T. and Roosma, J.: 1996, Global optimization with a limited solution time, *Journal of Global Optimization* **8**, 413–427.

Karnopp, D. C.: 1963, Random search techniques for optimization problems, *Automatica* **1**, 111–121.

Romeijn, H. E.: 1992, *Global Optimization by Random Walk Sampling Methods*, PhD thesis, Erasmus University Rotterdam, Rotterdam.

Scales, L.: 1985, *Introduction to Non-Linear Opimization,*, Macmillan, London.

Shubert, B.: 1972, A sequential method seeking the global maximum of a function, *SIAM Journal of Numerical Analysis* **9**, 379–388.

Törn, A. and Zilinskas, A.: 1989, *Global Optimization*, Vol. 350 of *Lecture Notes in Computer Science*, Springer, Berlin.

Zabinsky, Z. B. and Smith, R. L.: 1992, Pure adaptive search in global optimization, *Mathematical Programming* **53**, 323–338.

Zhigljavsky, A. A.: 1991, *Theory of global random search*, Kluwer, Dordrecht.