Global optimization algorithms for bound constrained problems

PhD Thesis

László Pál

University of Szeged
PhD School in Computer Science
Szeged, 2010

Global optimization algorithms for bound constrained problems

PhD Thesis

László Pál

Supervisor:
Prof. Dr. Tibor Csendes

University of Szeged
PhD School in Computer Science
Szeged, 2010

Acknowledgements

I would like to express my gratitude to my supervisor, Tibor Csendes, who has introduced me to the theory of global optimization, and who has provided me guidance and support. I would also like to thank him for his suggestions and solution proposals during this research. His advice and enthusiasm have taught me the balance between careful research and exploratory investigation.

I thank the Sapientia Institute for Research Programs and the Department of Mathematics and Informatics for supporting me throughout my doctoral studies.

During the development of this dissertation I also received support of my advisor's grants Aktion Österreich-Ungarn 70ou1 and TÁMOP-4.2.2/08/1/2008-0008 of the Hungarian National Development Agency.

Finally I want to thank my family. The encouragement and support from my wife Gabriella and our children Dániel was a powerful source of inspiration and energy.

Contents

Chapter 1

Introduction

Optimization problems arise in numerous fields, e.g. in operational research, engineering and science. In many problems, convexity of the objective function or the feasible domain cannot be easily verified, and it is reasonable to assume that several local optima exist. The classical theory of optimization as well as the standard techniques of modern mathematical programming, are aimed at finding only local optima. These techniques cannot generate or even use the global information needed to find the global minimum for a function with multiple local minima. In general, the classical optimization methods have difficulties in dealing with global optimization problems. One of the main reasons of their failure is that they can easily be entrapped in local minima.

Global optimization is a multidisciplinary research field and a branch of applied mathematics and numerical analysis that deals with the task of finding the absolutely best set to satisfy certain criteria, formulated in mathematical terms. In other words, it refers to the characterization and computation of the global extrema of a given nonconvex function in a certain feasible region which may have several local minimizers. Global optimization problems are typically quite difficult to solve exactly since many of them belong to the class of NP-complete problems. On the other hand, many real-world applications can be formulated as a global optimization problem. Hence, finding the optimum solution of such a problem is an important challenge.

Global optimization includes nonlinear, stochastic and combinatorial programming, multiobjective programming, control, games, geometry, approximation, algorithms for parallel architectures and so on. Global optimization has a wide range of applications. Some of these are applications like engineering design, production management, computational chemistry, product mixture design, environmental pollution management, parameter estimation, VLSI design, neural network learning, etc. Due to the rapid development of practical global optimization techniques in the last thirty years, it has gained the attention of researchers and practitioners from a lot of scientific domains like applied mathematics, operations research, industrial engineering, management science, computer science, and so on.

In this thesis, two important fields of continuous global optimization have been considered: the stochastic and the interval arithmetic based global optimization.

1.1 Outline of the dissertation

In this section, we summarize the organization of the thesis and we give brief descriptions of the main contributions done in this study.

Chapter 1 presents the basic definitions used throughout this thesis. A classification of global optimization problems with a literature survey is also given.

Chapter 2 deals with stochastic global optimization methods. The chapter begins with a review of the most important stochastic methods like the random search methods and two-phase methods. Section 2.3 is about the GLOBAL optimization method which is the basis of our studies in the subsequent sections. First, we present the original algorithm which is based on Bonder's stochastic clustering method. After that, we propose an improved version of the GLOBAL method which includes important changes. Our aim was to make some revisions and updates on the original algorithm in order to reach more efficiency and to improve its reliability. The experimental results on well-known test functions are shown in Section 2.4 to demonstrate the efficiency and reliability of the new GLOBAL method. We have completed three numerical tests. The first aimed to show the efficiency and reliability changes compared to the old version, while the second one was to compare the new method to C-GRASP, a greedy adaptive search technique. We also evaluated the performance of the GLOBAL algorithm on the BBOB-2009 noiseless testbed that contains problems which reflect the typical difficulties arising in real-world applications.

We conclude this chapter with an application from the domain of pension system. We analyzed the problem of designing a flexible pension scheme. Although the mathematical model of the problem contains constraints, we managed to solve it by GLOBAL using the penalty function method. Numerical results on the problem are also reported.

Chapter 3 deals with interval global optimization problems. The chapter begins with the basic concepts of interval analysis. We review the most important definitions of interval arithmetic and its properties. Next, we take a look at the inclusion functions and their properties. These serve as the main tool in the interval based optimization problems. In Section 3.3, different interval based branch and bound algorithms are presented. Our aim was to provide an easy to use reliable global optimization method using MATLAB. The section begins with the description of the most important interval techniques used by the interval branch and bound methods. In Subsection 3.3.2, we describe a basic algorithm for the bound constrained global optimization problem implemented in MATLAB using the INTLAB package. Our aim was to implement an algorithm variant that does not assume the differentiability of the objective function. The performance of the algorithm is reported after extensive numerical experiments on some well known functions by comparing it to the similar C-XSC based method. In Subsection 3.3.3, we present a more advanced algorithm which applies the common accelerating devices: the cutoff test, the concavity test, the monotonicity test, and the interval Newton step. The performance of the algorithm is reported based on extensive numerical experiments on some well known functions comparing it with the similar C-XSC based method. Subsection 3.3.4 describes an algorithm using a new condition for applying the Newton step. We have completed a computational test in order to compare the new condition with the old one. In Subsection 3.3.5, we investigate the theoretical aspects of the Newton step. More precisely, we analyze those cases when the Newton step is not successful.

We finalize this chapter with an application from the domain of sensor networks, namely the localization problem of sensor networks. We present some solution approaches including the interval based technique, too.

Chapter 4 gives a brief summary and conclusion of the main contributions in the thesis. The bound constrained test problems used throughout the study are given in Appendix A.

1.2 Basic definitions

Many recent problems in science, engineering and economics can be expressed as computing globally optimal solutions.

In general, the optimization problem is formulated in terms of finding the point x in a set X (called the feasible region) where a certain function $f : X \to \mathbb{R}$ (called the objective function), attains a minimum or a maximum. Without loss of generality, assume that the objective function is to be minimized. Global maximization problems can also be covered, since

$$\max_{x \in X} f(x) = \min_{x \in X}(-f(x)).$$

In this thesis, we will discuss methods for solving general global optimization problems (GOP) that can be formulated in the following way:

$$\begin{aligned} \min \quad & f(x), \\ \text{subject to} \quad & x \in X, \end{aligned} \tag{1.1}$$

where $f : X \to \mathbb{R}$ is the objective function, and $X \subseteq \mathbb{R}^n$ is the set of feasible points.

Definition 1.1. *A point x^* satisfying $f(x^*) \leq f(x)$ for all $x \in X$ is called a global minimizer of f over X, and the corresponding value of f is called a global minimum.*

The global minimum value of f on X is denoted by f^*, and the set of global minimizer points of f on X by X^*. That is,

$$f^* = \min_{x \in X} f(x) \qquad \text{and} \qquad X^* = \{x^* \mid f(x^*) = f^*\}.$$

Let $\|\cdot\|$ denote the Euclidean norm in $\mathbb{R}^n$ and let $\epsilon > 0$ be a real number. Then an $\epsilon -$ *neighborhood* of a point $y \in X$ is defined as

$$\mathcal{N}_\epsilon(y) = \{x \in X : \|x - y\| < \epsilon\}.$$

Definition 1.2. *A point $\overline{x} \in X$ is called a local minimizer of f over X, if there is an $\epsilon > 0$ such that*

$$f(\overline{x}) \leq f(x), \ \forall x \in \mathcal{N}_\epsilon(\overline{x}) \cap X.$$

Frequently, X^* consists only of a single point x^*, but this is not necessarily the case in general. For the sake of algorithmic tractability, we suppose throughout this study that X^* is countable. If the feasible set is given by its lower and upper bound,

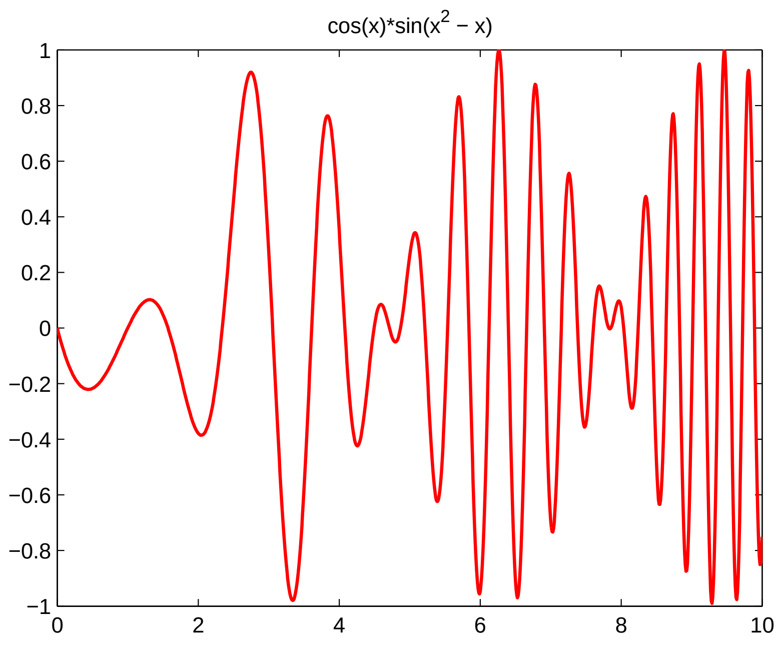


Figure 1.1: The one-dimensional, box-constrained model.

the problem (1.1) is called *bound constrained optimization problem.* This model is also called as *box-constrained optimization problem.*

Although in this thesis we restrict ourself mainly to studying algorithms for solving the problem (1.1), we also investigate applications which belong to a more general class of optimization problems formulated as

$$\begin{aligned} \min \quad & f(x), & & \\ \text{subject to} \quad & g_i(x) \leq 0, & & i = 1, 2, \ldots, m, \\ & x \in X, & & \end{aligned} \tag{1.2}$$

where $g_i(x) : \mathbb{R}^n \to \mathbb{R}$ are nonlinear constraint functions.

The problems (1.1) and (1.2) aim to find the minimizer points and the minimum values. This means we find the absolute minimal function value on the set of feasibility. This task is hard and it is hit by 'curse of dimensionality' which refers to the exponential increase of the volume of the search space as the dimension increases. We have illustrated this in Figures 1.1 and 1.2.

Definition 1.3. *The level set of f is defined by*

$$L(y) = \{x \in X \mid f(x) \leq y\},$$

and $L_x(y)$ denote the connected component of $L(y)$ containing x.

It is well known that the GOP is inherently unsolvable in a finite number of steps (see Dixon [37]) and in numerous cases, it is very difficult to solve exactly. Consequently, the more practical aim of GOP is to find suitable approximation of the global optima.

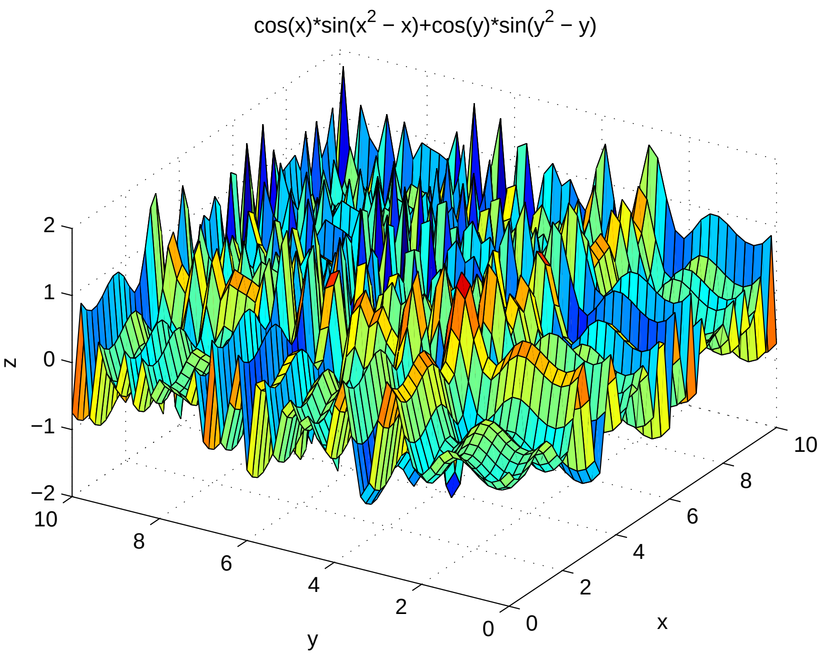


Figure 1.2: The two-dimensional, box-constrained model.

Usually we will therefore consider the global optimization problem solved if we have found a point in

$$\mathcal{N}_\epsilon(X^*) = \{x \in X : \|x - x^*\| < \epsilon \text{ for some } x^* \in X^*\}$$

or in the level set

$$X_\epsilon \equiv \{x \in X : f(x) \leq f^* + \epsilon\}$$

for some $\epsilon > 0$.

The previously introduced model (1.1) is very general: it includes linear programming, convex nonlinear programming models under corresponding additional assumptions. That is, there exists a considerable variety of specifications of the general GOP (1.1) which may differ in their additional analytical assumptions, related to the structure of X and f. In the subsequent sections we will specify the additional assumptions in the GOP definitions.

1.3 Classification of global optimization problems

The field of GOP can be subdivided into many categories according to possible formulations of the problem. We may classify for example with respect to the type of optimization decisions, to the type of objective function and constraints. During the last years the classification of optimization problems were considered by several authors and books. According to the *Handbook of Global Optimization*, edited by Horst and Pardalos (see Horst and Pardalos [57]), the global optimization techniques can be classified into two main categories: *exact* and *heuristic* methods.

Deterministic methods belonging to the first group, at least in theory, have a rigorous guarantee for finding at least one, or all global solutions. However, frequently the computational effort required by these methods increases exponentially with the dimension of the problem. That is, it is more preferable to use some stochastic techniques especially in higher dimensional problems. These procedures guarantee with probability one that the global optimum will be found. We should mention here that for many global optimization methods, the distinction between deterministic and stochastic is ambiguous. There are very effective methods, which involve deterministic as well as stochastic elements.

Heuristic methods do not possess convergence guarantees similar to those of exact methods. In the same time, they may provide good quality approximated solutions for many difficult GO problems, assuming that the method in question suits well the specific problem type.

A brief annotated classification of the most frequently applied GO strategies is provided below.

1.3.1 Exact methods

- **Branch and bound algorithms**. They consist of a systematic enumeration of all candidate solutions, where large subsets of candidates are discarded, by using upper and lower bounds of the objective function. The general branch and bound methodology is applicable to broad classes of global optimization problems, for example, in combinatorial optimization, concave minimization, reverse convex programs, DC programming, and Lipschitz optimization. Interval arithmetic based methods also belong to this category (see Ratschek and Rokne [95], Neumaier [76], Horst and Tuy [58], Kearfott [62], and Pintér [90]).

- **Enumerative strategies**. These methods are based on a complete enumeration of all possible solutions. Examples are vertex enumeration in concave minimization models, and generic dynamic programming in the context of combinatorial optimization (see Horst and Pardalos [57], and Horst and Tuy [58]).

- **Homotopy and trajectory methods**. These strategies have the objective of visiting all stationary points of the objective function. The method is applicable to smooth GO problems, but the computational requirement can be very high (see Horst and Pardalos [57]).

- **Naive approaches**. This category contains the grid search and pure random search methods. Although these methods are convergent under mild conditions, they have difficulties in solving higher dimensional problems (see Pintér [90]).

- **Stochastic search methods**. These procedures are based on random sampling, and their basic scheme can be improved by adding different enhancements like adaptive search strategy adjustments, sample clustering, and statistical stopping rules. Such methods are adaptive random searches, multistart methods, and Bayesian search strategies. These algorithms are applicable to both discrete and continuous GOPs under very general conditions (see Guss et al. [45], Horst and Pardalos [57], Mockus et al. [70], Pintér [90], and Zabinsky [116]).

1.3.2 Heuristic methods

- **Convex underestimation**. This strategy attempts to estimate the convexity characteristics of the objective function based on directed sampling. Convex underestimation strategies are applicable to smooth GOPs (see Dill et al. [36]).
- **Genetic algorithms, evolution strategies**. These methods emulate specific genetic operations (selection, crossover, and mutation) as these are observed in nature. These strategies are applicable to both discrete and continuous GO problems under mild structural requirements (see Michalewicz [69], Osman and Kelly [80], Glover and Laguna [43], and Voss et al. [114]).
- **Simulated annealing**. These techniques are based on the physical analogy of cooling crystal structures that spontaneously arrive at a stable configuration, characterized by – globally or locally – minimal potential energy. Simulated annealing is applicable to both discrete and continuous GOPs under mild structural requirements (see Osman and Kelly [80], and Glover and Laguna [43]).
- **Tabu search**. This search forbids or penalizes search moves which take the solution, in the next few iterations, to points in the solution space that have been previously visited. Tabu search methodology has been primarily used to solve combinatorial optimization problems, but it can also be extended to handle continuous GOPs (see Osman and Kelly [80], Glover and Laguna [43], and Voss et al. [114]).
- **Tunneling strategies**. Attempts to sequentially find an improving sequence of local optima, by gradually modifying the objective function. These strategies are applicable to smooth GO problems (see Levy and Gomez [65]).

Chapter 2

Stochastic global optimization

2.1 Introduction

Consider the following specification of the general GOP (1.1):

$$\min_{x \in X} f(x), \tag{2.1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is a continuous real valued function, $X \subset \mathbb{R}^n$ is a compact set. Some of the methods we will describe require additional assumptions on the objective function f or the feasible region X. We will note them wherever necessary.

As no algorithm can solve a general, smooth global optimization problem with certainty in finite time, stochastic methods are of eminent importance in global optimization. These methods incorporate probabilistic (stochastic) elements, which means that the result of the method is a random variable. In this thesis, we consider algorithms which use random numbers to generate new trial points. Therefore, we will have to sacrifice the absolute guarantee of success. However, under mild conditions on the sampling distribution and f, the probability that an element of $\mathcal{N}_\epsilon(X^*)$ or X_ϵ is sampled approaches 1 as the sample size increases (see Solis and Wets [108]). If the sample points are drawn from a uniform distribution over X and if f is continuous, then an even stronger result holds: the sample point with lowest function value converges to a global minimizer point value with probability 1.

The chapter is based on the following papers: Csendes et al. [33], Pál et al. [86], and Pál and Csendes [82].

The first part of the chapter deals with stochastic global optimization methods. First, we review the most important methods. After that, we describe in detail the GLOBAL optimization method.

An improved version of GLOBAL is also presented. The computational results obtained by the new GLOBAL method are reported and compared to the old version and also to the C-GRASP method. The contribution of the present author was the modification and implementation of a new MATLAB version of the GLOBAL method. The new version contains the BFGS local search method, and the improved UNIRANDI algorithm with the capability of uniform distribution direction selection. The test environment setup and the computational experiments were the contribution of the present author, too. The discussion part of the numerical results were done by the first coauthor with the help of the present author.

We also evaluated the performance of the GLOBAL algorithm on the BBOB-2009 noiseless testbed, containing problems which reflect the typical difficulties arising in real-world applications. The contribution of the present author was the modification of GLOBAL method in order to fit to the BBOB-2009 test environment, the completed computational experiments and the discussion on the results. The GLOBAL was also compared on the BBOB-2009 noiseless testbed with other well known global optimization algorithms. The comparison results are not included in the present thesis. For more details see Pošík et al. [91].

At the end of the chapter, we present a real-world application with numerical results. The contribution of the present author was the completion of the numerical test, while the model simplification and discussion on the numerical results were a common work with the coauthor.

2.2 A brief literature review

The two most important classes of problems included in stochastic methods are the random search methods and two-phase methods.

2.2.1 Random Search Methods

The class of random search methods consists of algorithms which generate a sequence of points in the feasible region following some prespecified probability distribution. The most basic algorithms from this class proceed by generating points from a single probability distribution. Alternatively, the distribution, from which a point in the sequence is determined, can be updated adaptively.

2.2.1.1 Pure Random Search

Pure Random Search (PRS) (see Brooks [20]), also referred to as Monte Carlo search is the simplest stochastic method for solving global optimization problems. The main idea is to sample a sequence of independent, identically distributed random points from the feasible region, while keeping the track of the best objective function value achieved. The pure random search is described by Algorithm 1.

The proof of convergence to the global optimum for the method was shown by many authors (see Devroye [35], Baba [2], Solis and Wets [108], and Pintér [89]). However, pure random search methods are not efficient, especially for problems with higher dimensions, since in this case the number of iterations required to solve a problem increases fast.

2.2.1.2 Pure Adaptive Search and Adaptive Search Methods

The Pure Adaptive Search (PAS) algorithm differs from the PRS algorithm in that it forces improvement in each iteration. In other words, an iteration point is generated from the uniform distribution on the subset of points that are improving with respect

Algorithm 1. The Pure Random Search

```
function PS($f, X$)
    $k := 1; y_0 := \infty$
    repeat
        Generate $x \in X$ by uniform distribution
        if $f(x) < y_{k-1}$ then
            $y_k := f(x); x_k := x$
        else
            $y_k := y_{k-1}; x_k := x_{k-1}$
        end
        $k := k + 1$
    until The stopping rule is satisfied
    return $x_k, y_k$
```

to the previous iteration point. More formally, an iteration point x_{k+1} is generated uniformly distributed in $S_{k+1} := \{x \in X \mid f(x) < f(x_n)\}$.

The PAS algorithm has been introduced and analyzed in Patel et al. [81] for convex programming problems, and in Zabinsky and Smith [117] for more general global optimization problems. The theoretical results provide an upper bound on the expected number of iterations to achieve a solution arbitrarily close to the global optimum. In other words, the expected number of PAS iterations grows at most linearly in the dimension of the problem.

Unfortunately, in practice, we encounter the following difficulties: constructing the improving region, generating a point uniformly distributed in S_n. These problems can be avoided by, instead of generating uniform points in X, generating points from a nonuniform distribution that assigns greater probability to the improving region S_n.

The Adaptive Search (AS) is a generalization of the Pure Adaptive Search method, where the iteration points are generated from a sequence of *Boltzmann distributions*.

Pincus (see Pincus [88]) and Rubinstein (see Rubinstein [100]) suggested that for approximating the global optimum, points should be generated from the Boltzmann distribution π_T with density function:

$$g_T(x) \propto e^{-f(x)/T},$$

where T is a positive number. This is appropriate because for small T the distribution π_T will concentrate near the global minimum.

The pseudocode of the AS method is presented by Algorithm 2, where τ is an $\mathbb{R}^+$-valued nonincreasing function.

An important advantage of this algorithm is that sampling is done from the feasible region X, instead of from a nested set of smaller level sets of f. This avoids the two difficulties of PAS. If we choose $T_k = \infty$ for all k, then the AS algorithm reduces to the PAS algorithm. However, the number of trial points necessary in line 5 can be influenced by an appropriate choice of the parameter T (line 9). The parameter T is called *temperature* parameter. A particular choice of temperature parameters $\{T_k\}_{k=0}^{\infty}$ is then called a *cooling schedule*.

Algorithm 2. The Adaptive Search Method

```
1.  function AS($f, X$)
2.    $k := 0; y_0 := \infty; T_0 := \infty$
3.    repeat
4.        repeat
5.            Generate $x$ from the distribution $\pi_{T_k}$ over $X$
6.        until $f(x) < y_k$
7.        $x_{k+1} := x$
8.        $y_{k+1} := f(x_{k+1})$
9.        $T_{k+1} := \tau(y_{k+1})$
10.       $k := k + 1$
11.   until The stopping rule is satisfied
12.   return $x_k, y_k$
```

The Adaptive Search preserves the linearity result of the Pure Adaptive Search algorithm, moreover the number of iterations is stochastically less than the number of iterations of PAS. As in the case of PAS, in practice, it is extremely difficult to generate points directly from the distribution π_T. That is, a lot of attention has been focused on finding algorithms to efficiently generate arbitrary distributions, using the Markov chain approach. Examples of such methods are *Hit-and-Run algorithms* (see Berbee et al. [8], Bélisle et al. [12]), and *Shake-and-Bake algorithms* (see Boender et al. [14]).

2.2.2 Two-phase methods

Many stochastic strategies in global optimization consist of two phases: the global phase and the local phase. During the global phase, random points are drawn from the domain of searches X according to a certain, often uniform, distribution. Then, the objective function is evaluated in these points. During the local phase, the sample points are manipulated by means of local search to yield a candidate global minimum. We assume that a proper local optimization method *Loc* exists. It can be started from an arbitrary point $x_0 \in X$ and then it generates the sequence of points in X which always converges to some $x^* := Loc(x_0) \in X$, that is the local minimizer attainable from the starting point x_0.

These methods are also called *Multistart* techniques because they apply local searches to each point in a random sample drawn from the feasible region (see Boender et al. [18], and Rinnooy Kan and Timmer [98, 99]). However, the Multistart method is inefficient in that it performs local searches starting from all sample points. That is some local minimizer points will be found several times. Since local searches are the most time consuming part of the method, it should ideally invoked no more than once in every region of attraction.

Various improvements were proposed by authors in order to reduce the number of local searches (see Rinnooy Kan and Timmer [98], and Törn [113]). The two most important methods which are aimed at reducing the performed local search number are: *clustering* methods and *Multi Level Single Linkage (MLSL)* algorithms.

Algorithm 3. The basic framework of the clustering methods

Step 1: Draw N points with uniform distribution in X, and add them to the current cumulative sample C. Construct the transformed sample T by taking the γ percent of the points in C with the lowest function value.

Step 2: Apply the clustering procedure to T.

Step 3: Apply the local search procedure to the points in T not yet clustered. Repeat Step 3 until every point has been assigned to a cluster.

Step 4: A stopping rule decides wether to return to Step 1 or to stop. If the method is stopped, then the local minimum found with the smallest function value is the candidate solution.

2.2.2.1 Clustering methods

The basic idea behind clustering methods is to form groups (clusters) of points around the local minimizers from a uniform sampled domain and start local searches no more than once in each of those groups. In other words, the procedure tries to identify the *regions of attraction* of the given function. The region of attraction of a local minimum x^* is the set of points in X starting from which a given local search procedure converges to x^*.

Two ways have been proposed how these groups can be created from the initial sample: *reduction* and *concentration*.

By reduction these groups usually are formed from the initial sample by taking a given percent of the points with the lowest function value (see Becker and Lago [7]). The set of remaining points, called a reduced sample, naturally approximate a level set of the objective function.

Concentration (see Törn [112]) consists of starting one or at most a few steepest descent steps from every point.

Clustering methods use mostly the reduction phase because the concentration may lead to failure. The sample, the resulting groups of points, may contain several regions of attraction, so that the global minimum can be missed. Or, one region of attraction may be divided over several clusters, in which case the corresponding minimum will be located many times.

The basic framework of the clustering methods is described in Algorithm 3. Clusters are formed stepwise, starting from a *seed* point, which may be an unclustered point with the lowest function value or the local minimum found by applying local search to this point. A point is added to the cluster using a *clustering rule*. That is why it is important to provide an appropriate rule to handle cluster growing and termination around a seed point. These rules usually use local information on the objective function and rely on properties of the sampling distribution.

The two most important clustering procedures are: *Density Clustering (DC)* (see Törn [112]), and *Single Linkage Clustering (SL)* (see Boender et al. [18], and Rinnooy Kan and Timmer [98]). The common scheme for the two clustering methods is

Algorithm 4. The common scheme for the two clustering procedures

```
1.  function BasicClustering($f, X$)
2.    $k := 0; X^* := \emptyset; X^{(1)} := \emptyset$
3.    repeat
4.      $k := k + 1$
5.      Draw $N$ points according to the uniform distribution on $X$
6.      Construct the transformed (reduced) sample
7.      Clustering to $X^*$ and $X^{(1)}$
8.      while Not all points from the reduced sample have been assigned to a
        cluster do
9.        $x^{(1)}$ – unclustered point from the reduced sample with the smallest
          objective value
10.       $x^* := Loc(x^{(1)})$
11.       if $x^* \notin X^*$ then
12.         add $x^*$ to $X^*$
13.         choose $x^*$ as the next seed point
14.       else
15.         add $x^{(1)}$ to $X^{(1)}$
16.         choose $x^{(1)}$ as the next seed point
17.       end
18.     end
19.   until Some global stopping rule is satisfied
20.   return The smallest local minimum value found
```

described in Algorithm 4.

In line 2, the X^* and $X^{(1)}$ sets are initialized, where X^* is a set containing the local minimizer points that were found so far, while $X^{(1)}$ is a set containing sample points to which the local search procedure has been applied unsuccessfully in the sense that already know local minimizer was found again. The algorithm contains a main iteration loop and the steps from line 3 to line 19 will be repeated until some global stopping rule is satisfied. In line 5, N points are drown uniformly on X. In line 6, a reduced sample is constructed by taking those γkN points of the current sample that have the lowest function values. A clustering procedure is applied then to the transformed sample. The elements of X^* are first chosen as seed points, followed by the elements of $X^{(1)}$.

Between lines 8 and 18 we iterate over the unclustered points from the reduced sample and apply a local search procedure to them to find a local minimizer point x^*. If x^* is a new local minimizer point, then we add it to X^* (line 12) and choose it as the next seed point (line 13), otherwise we add $x^{(1)}$ to $X^{(1)}$ (line 15) and use it as the next seed point for local search (line 16). In line 20, the smallest local minimum value is returned.

The two clustering methods differ how they form the clusters and how they calculate the critical distance in order to assign a point to a cluster (line 7).

For the algorithmic tractability we suppose that the objective function is twice continuously differentiable.

Density Clustering

Based on the Density Clustering, a cluster will correspond to the points in a subset T of X of stepwise increasing volume. The cluster will be terminated if in a step no points are added to the cluster.

We assume that the seed point of the cluster is a local minimizer point x^*. As we mentioned earlier, the set T should ideally have the same shape as the region of attraction around x^*. It is difficult to characterize these regions in general and we use instead the level sets that contain x^*. This suggests to let T correspond to $L_{x^*}(y)$ for stepwise increasing values of y. The level set is also hard to construct, but since f is twice continuously differentiable, we can approximate these sets by the level sets $\widetilde{L}(y)$ around x^* that are defined by the second order approximation $\widetilde{f}$ of f around x^*:

$$\widetilde{f}(x) = f(x^*) + \frac{1}{2}(x - x^*)^\top H(x^*)(x - x^*),$$

where $H(x^*)$ is the Hessian of f in x^*. Hence, in step i let T_i be the set

$$\{x \in X \mid (x - x^*)^\top H(x^*)(x - x^*) \le r_i^2\},$$

for some r_i. In this approach T_i are ellipsoids. The value of r_i is calculated so that the probability of too early termination of the cluster in step i, decreases with increasing k. The too early termination means that a cluster terminates in step i, while there are still unclustered reduced sample points in the level set.

The critical distance r_i chosen in Boender et al. [18] is

$$r_i(x) = \frac{1}{\sqrt{\pi}} \left(i \cdot \Gamma(1 + \frac{n}{2}) \cdot |H(x^*)|^{1/2} \cdot m(S) \cdot (1 - \alpha^{1/(N-1)}) \right)^{1/n},$$

where Γ is the gamma function, n is the number of variables of the problem, $|H(x^*)|$ denotes the determinant of $H(x^*)$, $m(S)$ is the Lebeque measure of the set S, N is the total number of sampled points, and $\alpha \in (0, 1)$ is a parameter of the clustering procedure.

The critical distance r_i chosen in Rinnooy Kan and Timmer [98] is

$$r_i(x) = \frac{1}{\sqrt{\pi}} \left(i \cdot \Gamma(1 + \frac{n}{2}) \cdot |H(x^*)|^{1/2} \cdot m(S) \cdot \frac{\zeta \ln(kN)}{kN} \right)^{1/n},$$

where ζ is some positive constant. This value of r_i guarantees that the probability that the cluster is terminated incorrectly in step i, decreases polynomially with increasing k.

DC has the property of asymptotic correctness, i.e. it finds the global minimum with the probability equal to 1 when k increases to infinity, but it does not have the property of the asymptotic probabilistic guarantee of finding all local minimizers, because a cluster can cover more than one minimizer and some local minimizers may remain undiscovered.

The basic disadvantage of this version of DC is that the objective function is approximated by the square function and the attraction sets are approximated by ellipsoids. If the level set differs significantly from an ellipsoid, then the criterion concerning when to stop the cluster recognition process is incorrect.

A method which does not fix the shape of the clusters in advance is the Single Linkage Clustering method.

Single Linkage Clustering

In this method the clusters are formed sequentially and each of them is initiated by a seed point. The distance between two points x and x' in the neighborhood of the local minimum x^* is defined as

$$d(x, x') = ((x - x')^\top H(x^*)(x - x'))^{1/2}.$$

After a cluster C is initiated, we find an unclustered point x such that

$$d(x, C) = \min_{y \in C} \|x - y\|$$

is minimal. This point is then added to C, after which the procedure is repeated until d exceeds some critical value r_k. This critical distance will be chosen to depend on kN only so as to minimize the probabilities of two possible failures of the method: the probability that a local search is started, although the resulting minimum is known already, and the probability that no local search is started in a level set which contains reduced sample points. The experiments showed that the SL method approximates the level sets more accurately than the DC.

The theoretical aspects of the Single Linkage method were investigated in Rinnooy Kan and Timmer [98]. The main results are summarized in the following theorem:

Theorem 2.1. *(Rinnooy Kan and Timmer [98]) If the critical distance r_k is given by the formula*

$$r_k(x) = \frac{1}{\sqrt{\pi}} \left(\Gamma\left(1 + \frac{n}{2}\right) \cdot m(S) \cdot \frac{\zeta \ln(kN)}{kN} \right)^{1/n},$$

then:

- *for $\zeta > 2$ the probability that the local method will be started by SL in step k tends to zero with increasing k,*
- *for $\zeta > 4$ even if the sampling of points had been carried out endlessly, the total number of local searches started by the SL method would be finite with the probability equal to 1.*

These attractive properties theoretically prove the efficiency of the method in terms of number of local searches performed. However, SL does not have the property of a probabilistic guarantee of success in the sense of finding all local minimizers, because the level set may contain more than one region of attraction, so that some local minima may be missed.

SL like DC has the property of asymptotic correctness, i.e. it finds the global minimum with the probability equal to 1 when k increases to infinity.

Algorithm 5. The Multi Level Single Linkage algorithm

```
1.  function MLSL($f, X$)
2.      $k := 0; X^* := \emptyset$;
3.      repeat
4.          $k := k + 1$
5.          Draw $N$ points $x_{(k-1)N+1}, \dots, x_{kN}$ according to the uniform distribution
            on $X$
6.          $i := 1$
7.          while $i \leq kN$ do
8.              if NOT (there is such $j$ that $f(x_j) < f(x_i)$ and $\|x_j - x_i\| < r_k$)
                then
9.                  $x^* := Loc(x_i)$
10.                 $X^* := X^* \cup \{x^*\}$
11.             end
12.             $i := i + 1$
13.         end
14.     until The global stopping rule is satisfied
15.     return The smallest local minimum value found
```

2.2.2.2 Multi Level Single Linkage algorithms

Multi Level Single Linkage (MLSL) has been derived from clustering methods (see Rinnooy Kan and Timmer [99]). Unlike DC or SL it does not contain the reduction phase and the superfluous clustering is omitted altogether.

In this method the local search procedure is applied to every sample point, except if there is another sample point within some critical distance which has a lower function value. The goal of MLSL is to find all local minimizers. The algorithm steps are given in Algorithm 5.

It has been proved that the algorithm has good asymptotic properties: the asymptotic probabilistic correctness and probabilistic guarantee of finding all local minimizers. The asymptotic properties of MLSL are summarized in the Theorem 2.2:

Theorem 2.2. *(Rinnooy Kan and Timmer [99]) If the critical distance r_k is given by the formula*

$$r_k(x) = \frac{1}{\sqrt{\pi}} \left(\Gamma(1 + \frac{n}{2}) \cdot m(S) \cdot \frac{\zeta \ln(kN)}{kN} \right)^{1/n},$$

then:

- *if $\zeta > 0$, then for any sample point $x \in X$ the probability that the local method will start from this point in step k decreases to zero while k increases,*
- *if $\zeta > 2$, then the probability that the local method will start in step k decreases to zero while k increases,*
- *for $\zeta > 4$ even if sampling is performed infinitely, the total number of local searches ever started by MLSL is finite with the probability 1.*

Similar to SL and DC, MLSL also has the property of asymptotic correctness, i.e it finds the global minimum with probability 1 with k increasing to infinity and unlike DC and SL, it has the property of probabilistic guarantee of success (in the sense of finding all local minimizers).

MLSL can be ineffective when the objective function has a large number of local minimizers which are close to each other and have small basins of attraction.

2.2.2.3 Stopping rules

As in the case of deterministic methods, one of the questions in applying a stochastic method is when to stop. Preferably, a method of this nature should terminate with some probabilistic information on the quality of the proposed solution. Several approaches based on different assumptions about the properties of possible objective functions f and using different stochastic techniques have been proposed to design a proper stopping rule. A Bayesian stopping rule for the Multistart algorithm has been introduced in Zieliński [119] and further developed in Boender and Zieliński [19], Boender and Rinnooy Kan [15, 16, 17], Betrò and Schoen [10, 11], and others. A summary of the most used stopping rules can be found in Schaefer [104].

Most Bayesian stopping rules for Multistart are based on the knowledge about the size of the sample and the number of local minimizers detected. The probabilistic model is estimated from information gathered during the run of the optimization procedure. This model can be used to construct the so-called *non-sequential stopping rules.* Information about the costs of further searches is not used in this approach. However, such information is used in the so-called *sequential rules* which will also be described in this chapter. Such stopping rules can be applied to every method, which if it starts from the same set of points as Multistart gives the same set of local minimizers.

Let k be the number of local minima of the objective function f, and let us denote the relative volume of the i-th region of attraction by $\theta_i,\ i = 1, \dots, k$. If these values would be known, then we can have several good stopping rules. In practice, $k, \theta_1, \dots, \theta_k$ are mostly unknown.

In the Bayesian approach the unknowns $k, \theta_1, \dots, \theta_k$ are assumed to be themselves random variables $K, \Theta_1, \dots, \Theta_K$ with realizations $k, \theta_1, \dots, \theta_k$, for which a prior distribution can be specified. Given the outcome $\{n_1, \dots, n_w\}$ of number of local searches, we then use Bayes' Theorem to compute the posterior distribution of $K, \Theta_1, \dots, \Theta_K$, which incorporates both our prior beliefs and the sample information. We assume that for the value of w every positive integer is equally probable, moreover, $\Theta_1, \dots, \Theta_K$ have the uniform distribution on the $(w-1)$-dimensional unit simplex.

If w different local maxima have been found in n independent searches, then two main results are that, the Bayesian estimate of the expected number of unobserved local maxima $(K - w)$, and the total size $(1 - \Omega)$ of unobserved regions of attraction are equal to

$$\begin{aligned} E(K-w) &= \frac{w(w-1)}{n-w-2}, \\ E(1-\Omega) &= \frac{w(w+1)}{n(n-1)}. \end{aligned}$$

The (a posteriori) probability that all local minimizers have been found is equal to:

$$\Pr(K = w) = \prod_{i=1}^{w} \left(\frac{n - 1 - i}{n - 1 + i} \right).$$

Non-sequential rules

The presented formulas can be applied to determine the following non-sequential stopping rules to terminate the search:

- Stop if the round-off of the estimated number of unobserved local minima is equal to 0, i.e.

$$\left[\frac{w(w+1)}{n - w - 2} \right] = 0. \tag{2.2}$$

- Stop if the estimated coverage of the sample of unobserved local minima is lower than an arbitrary constant τ_1, i.e.

$$\frac{w(w+1)}{n(n-1)} < \tau_1. \tag{2.3}$$

- Stop if the probability of finding all local minimizers is greater than an arbitrary constant τ_2, i.e.

$$\prod_{i=1}^{w} \left(\frac{n - 1 - i}{n - 1 + i} \right) > \tau_2. \tag{2.4}$$

Sequential rules

Sequential stopping rules take into consideration the cost of sampling. Let us consider the two following loss functions (see Guss et al. [45]):

$$L_1 = c_1^T \cdot (K - w) + c_1^E \cdot n,$$

and

$$L_2 = c_2^T \cdot (1 - \Omega) + c_2^E \cdot n.$$

Coefficients $c_1^T, c_2^T > 0$ are related to the cost of stopping the algorithm before all local minimizers are found. This cost is called a *termination loss*. Coefficients $c_1^E, c_2^E > 0$ are related to the cost of continuation of the search. It is called an *execution loss*.

Under the assumption that n points give w different minimizers, the expected a posteriori value of L_1 and L_2 (the so-called expected posterior loss or just the posterior loss) are equal to:

$$E(L_1 \mid (n, w)) = c_1^T \cdot \frac{w(w+1)}{n - w - 2} + c_1^E \cdot n, \tag{2.5}$$

$$E(L_2 \mid (n,w)) = c_2^T \cdot \frac{w(w+1)}{n(n-1)} + c_2^E \cdot n. \tag{2.6}$$

The purpose now is to find the optimal stopping rules which determine to stop the search when the current posterior loss is less than the expected posterior loss of carrying out another local search, given that the optimal strategy is adopted thereafter (see DeGroot [34]). For such an approach to be feasible we have to find a value n^* such that for all outcomes with $n > n^*$ the expected posterior loss of another search is always greater than the current posterior loss. To obtain such a result, we observe that given w different local maxima have been found in n searches, only two outcomes can occur as the result of another trial: either a yet unobserved local maximum is found, or not. Hence the conditional expected value of the posterior loss of one more step of the algorithm can be estimated on the basis of (2.5) and (2.6) according to the recurrent formula ($i = 1, 2$):

$$\begin{aligned} E(E(L_i \mid (n+1, W)) \mid (n,w)) = (1 - \frac{w(w+1)}{n(n-1)}) \cdot E(L_i \mid (n+1, w)) \\ + \frac{w(w+1)}{n(n-1)} \cdot E(L_i \mid (n+1, W+1)). \end{aligned} \tag{2.7}$$

From (2.7) we can determine that for L_1 the expected posterior loss of one additional local search minus the current posterior loss is always equal to

$$\delta = c_1^E - c_1^T \cdot \frac{w(w+1)}{n(n-1)}.$$

In this case there exists no n^* such that this value is positive for all $n \geq n^*$. However, a suboptimal Bayesian one-step-look-ahead stopping rule can be determined:

– Stop if
$$c_1^E - c_1^T \cdot \frac{w(w+1)}{n(n-1)} > 0.$$

For the second loss structure n^* can be shown to be equal to $n^* = \frac{c_2^T}{3c_2E}$. Given this result the following optimal stopping rule can be determined:

– Stop the search for all outcomes with $n \geq n^*$, and determine the optimal strategy for all outcomes with $n < n^*$ by working backwards from $n = n^*$ through the recurrence relation (2.7).

Stopping rules that use values of the objective function

The previously presented model was extended by Piccioni and Ramponi (see Piccioni and Ramponi [87]), who have succeeded in taking into consideration the function values of the local minima. The stopping rules consider undiscovered minimizers in which the objective value is less than the smallest value found earlier. The following stopping rules are analogous to rules (2.2), (2.3), and (2.4).

– Stop if the expected number of undiscovered better local minimizers is equal to zero, i.e.:

$$\left\lfloor \frac{w}{n-w-2} \right\rfloor = 0.$$

– Stop if the expected value of the sum of relative volumes of the basins of attraction of better local minimizers is less than a constant τ_1

$$\frac{w}{n(n-1)} < \tau_1.$$

– Stop if the probability that a better local minimizer does not exist is greater than a constant τ_2

$$\frac{n-w-1}{n-1} > \tau_2.$$

A suboptimal stopping rule can also be estimated for L_1 in a similar way (see Guss et al. [45]):

– Stop if

$$c_1^E - c_1^T \cdot \frac{w}{n(n-w-2)} > 0.$$

The value of n^* for the optimal stopping rule related to the loss function L_2 can be estimated from the formula:

$$n^* = \sqrt{\frac{c_2^T}{c_2^E}}.$$

2.3 The GLOBAL optimization method

Consider the following specification of the general GOP (1.1):

$$\min_{x \in X} f(x), \tag{2.8}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is a real valued function, $X = \{a_i \le x_i \le b_i,\ i = 1, 2, \ldots, n\}$ is the set of feasibility, an n-dimensional interval with lower and upper bounds of a and b, respectively. In general, we assume that the objective function is twice continuously differentiable, although it is not necessary for the global optimization framework procedure, and with a proper local search algorithm also nondifferentiable problems can be solved.

2.3.1 Boender's method

A popular stochastic global optimization algorithm was described in Boender et al. [18]. The algorithm structure is based on Algorithm 4, and used both clustering rules: Density Clustering (see Törn [112]) and Single Linkage Clustering (see Boender et al. [18], and Rinnooy Kan and Timmer [98]).

They also used the *gradient criterion*: in both clustering procedures, before a point x is added to a cluster, the negative gradient at x is verified to point in the direction of x^*. In other words, if the derivative of f in x in the direction of x^* is positive, then the x will be rejected. This step was necessary because often the level set contained various local minimizer points within one connected subset.

The method also constructs a range of confidence intervals on the true value of the global minimum using the two smallest function values found in the complete sample.

The algorithm was tested on standard test functions and compared with other methods. The results showed that it was a reliable and computationally attractive method for global optimization.

2.3.2 The original GLOBAL algorithm

GLOBAL (see Csendes [27]) is a stochastic method based on Boender's algorithm (see Boender et al. [18]). In several recent comparative studies (e.g. Mongeau et al. [73], Moles et al. [72]), this method performed quite well in terms of both efficiency and robustness, obtaining the best results in many cases.

Its goal is to find all local minimizer points that are potentially global. These local minimizers will be found by means of a local search procedure, starting from appropriately chosen points from the sample drawn uniformly within the set of feasibility. In an effort to identify the region of attraction of a local minimum, the procedure invokes a clustering procedure. The main algorithm steps are based on Algorithm 4 and is described in Algorithm 6.

Although GLOBAL is based on Boender's method, it contains several modifications and improvements. The most important changes are:

- The Single Linkage Clustering was selected after a respective testing.
- The clustering distance is not based on the Hessian (thus the latter should not be computed).
- The gradient criterion for forming clusters has been found to be less effective and it is left out.
- No steepest descent step is used to transform the original sample.
- The less informative confidence intervals are not calculated for the global minimum value.
- A scaling of the original problem is applied to ensure better numerical stability.

Algorithm 6. The GLOBAL algorithm

```
function GLOBAL($f, X$)
    $k := 0; X^* := \emptyset; X^{(1)} := \emptyset$
    repeat
        $k := k + 1$
        Generate $N$ points $x_{(k-1)N+1}, \ldots, x_{kN}$ according to the uniform
        distribution on $X$
        Determine the reduced sample consisting of the $\gamma kN$ best points from the
        sample $x_1, \ldots, x_{kN}$
        Clustering to $X^*$ and $X^{(1)}$
        while Not all points from the reduced sample have been assigned to a
        cluster do
            $x^{(1)}$ – unclustered point from the reduced sample with the smallest
            objective value
            $x^* := Loc(x^{(1)})$
            if $x^* \notin X^*$ then
                $X^* := X^* \cup \{x^*\}$
                choose $x_s := x^*$ as the next seed point
            else
                $X^{(1)} := X^{(1)} \cup \{x^{(1)}\}$
                choose $x_s := x^{(1)}$ as the next seed point
            end
            Add all unclustered reduced sample points which are within
            distance $r_k$ of a point already in the cluster initiated by the seed
            point $x_s$
        end
    until Some global stopping rule is satisfied
    return The smallest local minimum value found
```

Apart from the above changes, the original GLOBAL algorithm includes two different local search procedures: a quasi-Newton procedure with the Davidon-Fletcher-Powell (DFP) update formula (see Gill et al. [42]) and a random walk type direct search method UNIRANDI (see Järvi [60]).

The quasi-Newton method applies numerical derivatives calculated inside of it, so the user must not include subroutines for the calculation of derivatives, only that for the objective function itself. In this sense GLOBAL is a direct search method. According to our experience, it is a quite important feature, that allows a wide range of applications. With advanced quasi-Newton type local search methods a similar good convergence speed can be achieved as with local search methods applying first and second order derivatives. Since the algorithm uses only objective function evaluations on places determined by the algorithm, the black-box type objective function can also be defined by only a procedure, without having an explicit expression for it. On the other hand – if necessary – sophisticated automatic differentiation tools enable us to calculate the necessary derivatives without much human interaction.

The GLOBAL method has its own stopping criteria that stops if in an iteration no

Algorithm 7. The UNIRANDI local search method

```
1.  function UNIRANDI($f, x$)
2.    $i := 0$
3.    while The convergence criterion is not satisfied do
4.        Generate a unit length random direction $d$
5.        Find a trial point $x_{trial} := x + h \cdot d$
6.        if $f(x_{trial}) < f(x)$ then
7.            $x := LineSearch(f, x_{trial}, x, d, h)$
8.            $h := 0.5 \cdot h$
9.            continue
10.       end
11.       $d := -d$
12.       $x_{trial} := x + h \cdot d$
13.       if $f(x_{trial}) < f(x)$ then
14.           $x := LineSearch(f, x_{trial}, x, d, h)$
15.           $h := 0.5 \cdot h$
16.           continue
17.       end
18.       $i := i + 1$
19.       if $i < 2$ then continue
20.       $h := 0.5 \cdot h$
21.       $i := 0$
22.   end
23.   return $x$
```

new local minimizer point was found.

The UNIRANDI procedure is a random walk type robust local search method, which can be used when the problem structure does not allow us to utilize the locally quadratic behavior as it is the case for the quasi-Newton technique. The method consists of two main steps: generation of random directions and line searches along these directions. The local search algorithm is described in Algorithm 7.

Between line 3 and line 22, the algorithm iteratively generates random directions and makes line searches along them. At each iteration, in line 4, a unit random direction d is generated in the interval $[-0.5, 0.5]^n$, but they are accepted only if the norm is less or equal to 0.5. This condition means that points outside the hypersphere of radius 0.5 are discarded in order to obtain a uniform distribution of random directions.

After finding a proper direction in line 5, a trial point is computed. The parameter h that controls the step length, is initialized to 0.001. If the trial point has a smaller objective value than the current best solution, then, in line 7, we start a line search along the d direction starting from it. The line search algorithm is described in Algorithm 8. If we didn't find a better point along the current direction, we try the opposite direction in line 11. In line 20, we halve the step length if the algorithm couldn't improved the current best solution. After two unsuccessful step reduction, in line 19, we choose a new direction. The algorithm stops if the relative convergence of the objective function or the step length is less than 10^{-6}. The best solution found, over all iterations, is returned.

Algorithm 8. The LineSearch algorithm

```
1. function LineSearch($f, x_{trial}, x, d, h$)
2.   while $f(x_{trial} < f(x))$ do
3.       $x := x_{trial}$
4.       $h := 2 \cdot h$
5.       $x_{trial} := x + h \cdot d$
6.   end
7.   return $x$
```

The GLOBAL method was originally coded in Fortran and C languages and it was capable to solve global optimization problems of up to 15 variables. The efficiency and reliability of the procedure was tested on standard test functions. It had been applied also for theoretical chemical problems in Balogh et al. [4], and for optimization in abstract spaces, on Stiefel manifolds in Balogh et al. [3]. It was also used in real-life or industrial applications. We just mention some from the fields of bioprocess analysis (see Banga et al. [5]), climate control (see Moles et al. [71]), and integrated process design for wastewater treatment plants in Moles et al. [72].

Although the bound constrained problem definition (1.1) does not allow explicit non-linear constraints, the use of penalty functions enables us to cope also with constrained problems. According to our experiences, this approach can be successful in most of the cases encountered (see Csendes et al. [31]). There is also an extension of the algorithm to constrained problems in Sendín et al. [105].

2.3.3 The improved GLOBAL optimization method

The GLOBAL method has been introduced in the 1980s for bound constrained global optimization problems with black-box type objective function. Since then the technological environment has been changed much. Therefore, our aim was to make some revisions and updates on the involved algorithms to utilize the novel technologies, and to improve its reliability.

The main part of the work consists of experiments done with the GLOBAL algorithm. As a result of these tests, we obtained a more efficient and robust method. During these experiments we tried to solve the problems with parameter tuning. On the other hand, we observed that GLOBAL is sensible for the input parameters, hence we also tried to optimize them during the running of the algorithm. However, we find that this induces a substantial increase in the running time, which is not acceptable in our case.

The most important changes made on GLOBAL are:

- It is now coded in MATLAB, utilizing the vectorization technique for better efficiency.
- We use the Broyden-Fletcher-Goldfarb-Shanno (BFGS) (see Broyden [21]) local search algorithm instead of the earlier DFP method.
- Better uniform and normal distribution random number generators are applied.

Table 2.1: The number of points in percent outside the hypersphere of radius 0.5.

Dimension	2	3	5	7	10	12	15
Outside points (%)	21	45	83	96	99	99	100

– Some improvements were made in the uniform distribution direction selection procedure of the UNIRANDI local search method. The new code provide better statistical characteristic while it needs less computation. The present implementation of UNIRANDI works now without dimension related problems.

We have utilized the advantages of MATLAB to obtain an efficient code. The vectorization of MATLAB, a special syntax, makes it easy to obtain such a machine code that can feed the pipeline of the CPU in such a way that long vector calculations can achieve a closely full use of the processor pipeline. This is otherwise hard to reach by high level algorithmic languages. The last mentioned improvement in the list above had enhanced the efficiency of GLOBAL substantially in terms of CPU time used. Now we have the capability to solve larger problems than before with similar reliability.

The BFGS local search method works similarly like the DFP algorithm. The main difference is that the first one uses a different update formula. The comparison results (see Powell [92]) show that the quasi-Newton method with the BFGS update formula performs better than the one with the DFP update formula.

In the UNIRANDI local search method, the random directions are uniformly generated in the interval $[-0.5, 0.5]^n$, but they are accepted only if the norm is less or equal than 0.5. This condition means that points outside the hypersphere of radius 0.5 are discarded in order to obtain a uniform distribution of random directions (i.e. to avoid having more directions pointing toward the corners of the hypercube). As the number of variables increases, it becomes more difficult to produce points satisfying this condition and for more than 15 variables it is never satisfied. This can be observed in the Table 2.1, which contains the average number of points in percent outside the hypersphere of radius 0.5 in different dimensions. The results represent the average of 10 independent runs. In a single run, we sampled 100 points.

In order to fix this problem, we changed the UNIRANDI so that, the random directions will be generated by normal distribution $\mathcal{N}(0, 1)$, instead of uniform distribution and the respective vectors are normed.

We have illustrated two iterations (see Figure 2.2) of the GLOBAL algorithm using the UNIRANDI local search in the case of Branin function (see Figure 2.1 and Appendix A).

2.4 Numerical results

We have completed two sets of numerical tests: the first aimed to show the efficiency and reliability changes compared to the old version, based on the published results in Csendes [27], and one to compare the new method to C-GRASP, a greedy adaptive search technique (see Feo and Resende [40]) modified to solve continuous global optimization problems published in Hirsch et al. [55].

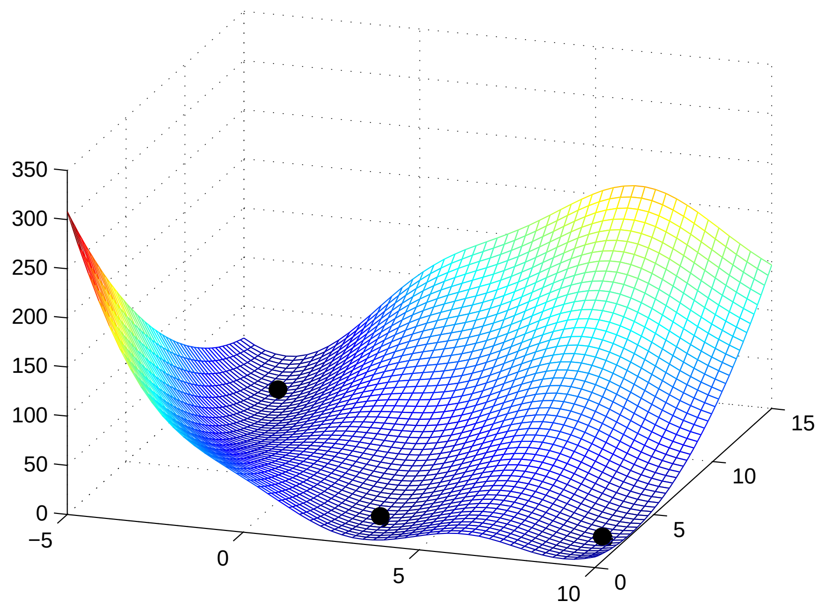
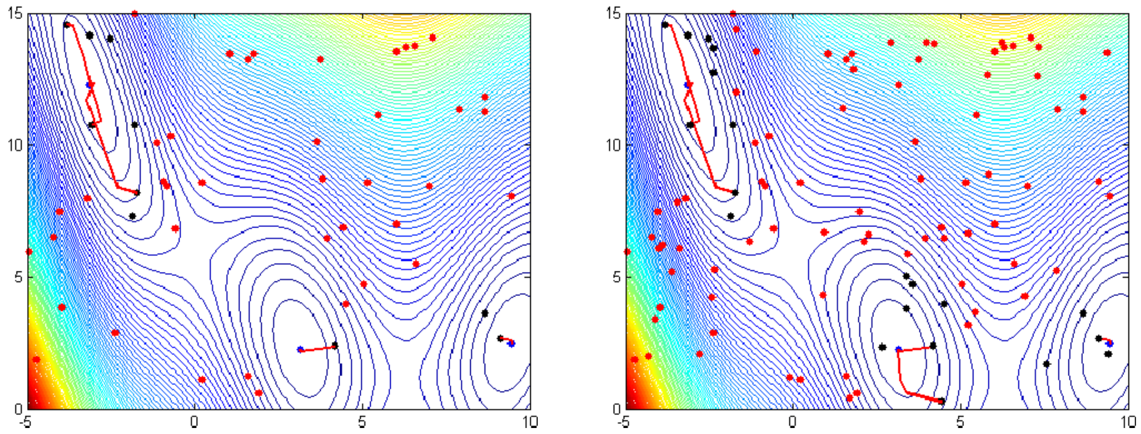


Figure 2.1: The Branin function with three global optima.



(a) First iteration: 50 points are drown uniformly (red points), 10 best points are clustered (black points).

(b) Second iteration: another 50 points are drown uniformly (red points), 20 best points are clustered.

Figure 2.2: Two iterations of the GLOBAL algorithm with the UNIRANDI local search.

Table 2.2: The number of function evaluations for the old and new versions of GLOBAL with the quasi-Newton local search routines on the standard test problems.

Problem		old	new				
	dim.	av.	av.	min.	max.	median	st. dev.
Shekel-5	4	990	1,090	540	2,235	1,067.5	352.52
Shekel-7	4	1,767	1,718	895	3,160	1,655.0	449.16
Shekel-10	4	2,396	2,378	1,145	4,630	2,317.5	697.19
Hartman-3	3	216	196	86	385	171.5	61.55
Hartman-6	6	1,446	703	132	1,647	702.5	263.32
Goldstein-Price	2	277	286	130	569	270.5	88.59
Branin	2	330	77	61	115	73.0	11.84
SHCB	2	233	107	58	224	103.5	35.67
Rosenbrock	2	410	125	52	334	109.0	62.27

Our numerical experiments were completed on a PC with a 3.0 GHz Pentium-4 processor and 1 Gbyte memory. We used the standard time unit (1,000 evaluations of the Shekel-5 function at $x^T = (4.0, 4.0, 4.0, 4.0)^T$) to measure the computation time comparably. GLOBAL has three parameters to set: the number of sample points (abbreviated in the tables as sample size, values between 20 and 100,000), the number of best points selected (denoted as selected, between 1 and 20), and the stopping criterion parameter named (precision, values between 4 and 8 digits).

2.4.1 Comparison with the old GLOBAL version

We used the standard test functions applied for the old version. For each problem we made 100 independent runs (earlier it was just 10), and we recorded the average number of function evaluations and the average CPU time necessary, measured in the standard time unit. The parameters of the procedures were set so that the algorithm was able to find the global optimum each time. In this sense the present algorithm was set to achieve about one order of magnitude better reliability. Here we understand reliability as the ability of an algorithm to find a global minimizer point a given number of times out of a preset number of independent runs. While in the case of the old GLOBAL only 10 successful runs were required out of 10 independent runs, now we tuned our algorithm to meet this criterion for 100 successful runs out of 100, which means the new one is certainly much more reliable. The good algorithm parameter values found (given in the tables) are telling also on how the procedure can achieve reliability. The criterion to accept an approximative solution as a successful estimate of the global minimizer was the same as in Csendes [27]: if the relative distance between them was less than 10^{-2}.

Table 2.2 provides the average number of function evaluations for the old and the new versions using the quasi-Newton type local search routines on the standard test problems. In case of the new implementation also the minimal, the maximal number of function evaluations are given together with the median and the standard deviation. According to the results, for most of the test functions the new algorithm was substantially better than the old one. It is so mostly just due to the more sophisticated local search technique used (BFGS instead of an old implementation of DFP). The reason for it can be seen

Table 2.3: The number of function evaluations for the old and new versions of GLOBAL with UNIRANDI as local search method on the standard test problems.

Problem		old	new				
	dim.	av.	av.	min.	max.	median	st. dev.
Shekel-5	4	1,083	1,450	751	2,718	1,374.0	413.88
Shekel-7	4	1,974	2,527	1,254	4,834	2,530.0	695.24
Shekel-10	4	2,689	3,429	1,636	5,795	3,277.5	881.97
Hartman-3	3	697	1,449	363	3,933	1,330.0	671.15
Hartman-6	6	2,610	2,614	274	9,004	2,132.5	1,676.16
Goldstein-Price	2	386	446	126	1,118	413.0	211.94
Branin	2	464	172	92	353	149.0	64.46
SHCB	2	267	176	95	328	175.0	66.85
Rosenbrock	2	1,524	1,081	277	2,687	902.0	516.38

Table 2.4: The average CPU times measured in standard time units for the old and new versions of GLOBAL with the quasi-Newton and UNIRANDI local search methods on the standard test problems. The algorithm parameters are the sample size, the number of selected points, and the precision required from the local search, respectively.

Problem	quasi-Newton					UNIRANDI				
	old	new				old	new			
	av.	av.	parameters			av.	av.	parameters		
Shekel-5	3.0	5.14	100	10	6	3.5	3.87	100	12	6
Shekel-7	4.9	7.89	200	15	6	6.0	6.64	300	15	6
Shekel-10	7.0	10.46	250	15	6	8.8	9.14	400	15	6
Hartman-3	1.2	1.19	15	2	7	1.9	3.86	15	3	7
Hartman-6	4.2	3.39	10	3	6	14.2	7.68	20	3	6
Goldstein-Price	1.3	1.32	50	4	6	1.5	0.81	30	4	7
Branin	1.4	0.40	20	1	6	1.6	0.31	20	1	6
SHCB	1.2	0.61	20	2	6	1.3	0.33	20	2	6
Rosenbrock	1.0	0.83	2	1	7	1.5	1.93	2	1	7

on Table 2.3 with basically the same local search procedure, the improvement in the average number of function evaluations appeared for only one third of the test functions. Yet even the figures for the new method with UNIRANDI are nice since the reliability improved much.

The algorithm parameters (shown in Table 2.4) cannot be compared in detail, since they were not collected and reported for the old method, still they are close to each other (as far as it can be judged). The fitting parameter setting is straightforward for the sample size and the number of selected points: the larger the sample size the more reliable the algorithm. The smaller the selected/sample size ratio, the smaller local minima we can find (i.e. the larger local minima will not be identified – the more efficient the search). It was one of the important findings of the papers Boender et al. [18], and Rinnooy Kan and Timmer [98, 99] that good reliability can be achieved efficiently while not locating all the local minimizer points. The finding of a local minimizer point with much worse function value than the global optimum does not contribute much for the identification of a global minimizer. This is what makes it possible to save computational efforts while keeping reliability in the determination of global minimizer points.

The set precision of the local search method has a different role: in general it should be close to the required relative precision of the estimated global minimum value. However, according to our experience, for both local search methods it is worth to set the precision value higher to avoid cases when a local minimum is not recognized since the results of local searches could not be identified as the same – although the same region of attraction was found. This phenomenon is reflected in Table 2.4 in the larger precision values for those test problems for which the objective function values do not change much close to a global minimizer point.

The average CPU times (measured in standard time units) necessary for the solution of the test problems are comprised in Table 2.4. The figures follow more or less the differences seen for the number of function evaluations in the previous tables. The differences to the anticipated ratios are due to overhead of startup and output in part independent of the problems, the different programming environments, hardware and even because of the way the standard time units were measured (now actually the time for the 10^6 evaluation of the Shekel-5 function was divided by 1,000).

The conclusion of the first set of tests completed is that on standard test problems the new implementation is closely as good in terms of efficiency as was the old one, while the reliability of the solution has been increased substantially. Due to the better quasi-Newton local search method, the new version is much better for smooth problems even in terms of the necessary number of objective function evaluations.

2.4.2 Comparison with the C-GRASP method

As it was already mentioned, the C-GRASP method extends the greedy randomized adaptive search procedure presented in Feo and Resende [40] from the domain of discrete optimization to that of continuous global optimization (see Hirsch et al. [55]). It does not make use of derivative information, thus it is a direct search method. Regarding its control structure, it is a multistart local search technique too, such as GLOBAL. Since its local search part does not utilize the possible smoothness of the objective function, it is fair only to compare it to GLOBAL with UNIRANDI.

The two computers used for the testing were of similar performance, but our one was slightly slower and had much less memory (however the latter has not affected much the comparison results). In this way the direct CPU times can only be used to compare with some care.

We applied our new implementation of GLOBAL to the same set of 14 global optimization test problems on which C-GRASP was run. The global minimum value f^* was known for all problems in the test set. Both methods were run until the objective function value f was significantly close to global optimum (i.e. till $|f^* - f| \leq 10^{-4}|f^*| + 10^{-6}$ became true). GLOBAL could also be stopped when no new local minimizer point was found in the last iteration cycle. The use of the known optimum value within an optimization algorithm is not typical, still it is realistic in several cases, as explained in Hirsch et al. [55], but also according to the experiments e.g. with circle packing problems (see Markót and Csendes [67], and Szabó et al. [110]).

Table 2.5: The number of function evaluations for C-GRASP and GLOBAL with the UNIRANDI local search routine. The algorithm parameters were: number of sample points: 400, selected points 15, and the required number of precise digits for the local search 8.

Problem	C-GRASP	GLOBAL with UNIRANDI				
	average	average	min.	max.	median	st. dev.
Shekel-5	5,545,982	1,489	897	2,259	1,470.0	298.56
Shekel-7	4,052,800	1,684	860	2,914	1,625.0	338.05
Shekel-10	4,701,358	1,815	1,034	3,623	1,713.5	532.08
Hartman-3	20,743	3,608	1,767	5,844	3,585.0	828.73
Hartman-6	79,685	16,933	6,240	29,665	16,739.5	5,040.56
Goldstein-Price	29	923	559	1,558	874.0	229.22
Branin	59,857	1,023	621	1,678	1,032.5	211.07
Rosenbrock-2	1,158,350	6,274	2,894	10,378	6,091.0	1,402.33
Rosenbrock-5	6,205,503	374,685	168,601	628,507	360,241.0	89,941.28
Rosenbrock-10	20,282,529	1,908,469	806,288	2,418,556	2,043,155.0	477,478.51
Easom	89,630	1,604	532	2,664	1,610.5	416.63
Shubert	82,363	1,399	936	1,859	1,407.5	171.34
Zakharov-5	959	8,227	4,629	11,420	8,367.5	1,702.57
Zakharov-10	3,607,653	47,288	34,549	53,995	47,850.0	3,767.57

For each problem, 100 independent runs of GLOBAL were completed. We recorded the percentage of runs that found a significantly close solution, the time necessary for such solutions and the number of function evaluations. The algorithm parameters of GLOBAL were set again such a way that it was able to find a global minimizer point in each run. We listed the results in the Tables 2.5 and 2.6.

Table 2.5 contains the numbers of objective function evaluations necessary to solve the test problems in the above described sense. According to it GLOBAL was with the exception of the Goldstein-Price and Zaharov-5 problems always much more efficient than C-GRASP, sometimes even by orders of magnitude. The reason for that may be due to the different model in the background: while C-GRASP is prepared for any ugly behavior, GLOBAL assumes inherently that at least one global minimizer points has a non-negligible sized region of attraction. Still, both algorithms are capable to solve black box problems, i.e. without additional information on the problems beyond the objective function value (when GLOBAL is used with UNIRANDI).

Note that for GLOBAL the algorithm parameters should be such that all problems could be solved all times it was run. In contrast to that, it can be surprising that some problems (e.g. Shekel-10) could be solved by the new GLOBAL with less function evaluations than what is stated in Table 2.2. This phenomenon is caused by the new stopping rule that can stop iteration earlier. It can also be seen that the higher dimensional versions of some test function could be solved by both techniques with correspondingly larger computational efforts.

The less comparable CPU time values are summarized in Table 2.6. These reflect on one hand the anticipated differences in the problem difficulty according to the number of function evaluations. On the other hand the CPU times for the Rosenbrock test functions are hard to explain: although they are proportional to the number of function

Table 2.6: CPU time required by C-GRASP and GLOBAL with the UNIRANDI local search routine in seconds.

Problem	dim.	C-GRASP	GLOBAL
Shekel-5	4	2.3316	0.1313
Shekel-7	4	2.3768	0.1461
Shekel-10	4	3.5172	0.1614
Hartman-3	3	0.0026	0.3208
Hartman-6	6	0.0140	1.7880
Goldstein-Price	2	0.0000	0.0516
Branin	2	0.0016	0.0580
Rosenbrock-2	2	0.0132	0.4117
Rosenbrock-5	5	1.7520	24.7559
Rosenbrock-10	10	11.4388	130.6813
Easom	2	0.0042	0.0916
Shubert	2	0.0078	0.0930
Zakharov-5	5	0.0000	0.5369
Zakharov-10	10	1.0346	3.1428

evaluations for GLOBAL, the efficiency relation between the two methods found earlier cannot be recognized here.

Summarizing the results we can conclude that the new version of GLOBAL utilizes the advantages offered by MATLAB, and the algorithmic improvements increased the size of the problems that can be solved reliably with it. The reliability of the algorithm is now better while the efficiency is improved, too. The careful comparison both with the old version and with C-GRASP is favorable for the new version of GLOBAL.

2.4.3 Benchmarking the GLOBAL on the BBOB-2009 Noiseless Testbed

The Black-Box Optimization Benchmarking (BBOB) 2009 was a benchmarking event that took place at a workshop in GECCO 2009, Montreal, Canada[1]. The aim of BBOB-2009 was to quantify and compare performance of optimization algorithms by providing a COCO software platform[2] as well as a full experimental set-up for participants. Using the platform we have the possibility to chose and implement a single-objective benchmark function testbed, to design of an experimental set-up, to generate data output for post-processing and presentation of the results in graphs and tables. The framework also provides two testbeds: noise-free and noisy functions.

2.4.3.1 The test environment description

The numerical experiments are conducted on a testbed comprising twenty-four noiseless test functions (see Finck et al. [41], Hansen et al. [53]). These functions have been

[1]http://www.sigevo.org/gecco-2009/workshops.html#bbob

[2]http://coco.gforge.inria.fr

constructed so that they reflect the real-world application difficulties and are split into several groups like separable functions, functions with low or moderate conditioning, functions with bad conditioning and unimodal, multimodal ones with adequate global structure, multimodal problems with weak global structure. All functions are scalable with the dimension, hence we can use 2, 3, 5, 10, 20, and 40 dimensional search space, respectively. Additionally, all functions have their optimum in the $[-5; 5]^n$ range, which can be a reasonable setting for the search domain. Each function is tested over five different instances. The experiments were repeated three times for each instance. The algorithm performance was evaluated over all 15 trials.

The success criterion of a run is to reach the $f_{target} = f_{opt} + \Delta f$ target value, where f_{opt} is the optimal function value, and Δf is the precision to reach.

In order to quantify the search cost of an algorithm, a performance measure should be provided. The main performance measure adopted is the runtime ERT, *Expected Running Time* (see Hansen et al. [50]; Price [94]). The ERT number depends on a given target function value, $f_{\mathrm{t}} = f_{\mathrm{opt}} + \Delta f$, and is computed over all relevant trials as the number of function evaluations used during the trials while the best function value did not reach f_{t}, summed over all trials and divided by the number of trials that actually reached f_{t}. In other words, The ERT value estimates the expected number of function evaluations for the target function value to be reached if the algorithm is restarted until a single success can be reached.

The results are also presented using the empirical cumulative distribution function of the distribution of ERT divided by n to reach a given target function value. It shows the empirical cumulated probability of success on the problems considered depending on the allocated budget.

For a more detailed environment and experimental description see in Hansen et al. [50, 51].

2.4.3.2 Parameter tuning and setup

In this study, GLOBAL has six parameters to set: the number of sample points set within an iteration step, the number of best points selected for the transformed sample, the stopping criterion parameter for the local search, the maximum number of function evaluations allowed for local search, the maximum number of local minima to be found, and the type of used local method. All these parameters have a default value and usually it is enough to change only the first three of them.

In all dimensions and for all functions we used 300 sample points, and the 2 best points were kept for the transformed sample. In 2, 3, and 5 dimensions we used the Nelder-Mead simplex method (see Nelder and Mead [75]) implemented in Kelley [63] as a local search with 10^{-8} as tolerance value and with 5,000 as the maximum number of function evaluations. In 10 and 20 dimensions for the f_3, f_4, f_7, f_{16}, f_{23} functions we used the previous settings with a local search tolerance of 10^{-9}. Finally, in the case of the remained functions we used the previous parameters with the MATLAB `fminunc` function as a local search method using the BFGS update formula and with 10,000 as the maximum number of function evaluations.

As it can be observed, during the parameter tuning we used two different settings. In lover dimensions we used the Nelder-Mead method while in higher dimensions the BFGS local search was applied except five functions. Although this kind of a priori parameter setting is not suggested, the two important parameters of GLOBAL (the number of sample points, the number of best points selected) were the same on the entire testbed. The different settings may be characterized with the entropy measure crafting effort (see Price [94], Hoos and Stützle [56]) for each dimensionality in the following way:

$$\mathrm{CrE} = -\sum_{k=1}^{K} \frac{n_k}{n} \ln\left(\frac{n_k}{n}\right),$$

where K is the number of different parameter settings used over the whole testbed, $n = \sum_{k=1}^{K} n_k$ is the number of functions in the testbed and n_k is the number of functions, where the parameter setting with index k was used for $k = 1, \ldots, K$.

The crafting effort calculated in case of our settings was (in dimensions 10 and 20): $\mathrm{CrE}_{10} = \mathrm{CrE}_{20} = -(\frac{5}{24}\ln\frac{5}{24} + \frac{19}{24}\ln\frac{19}{24}) = 0.5117$.

2.4.3.3 CPU timing experiment

For the timing experiment the GLOBAL algorithm was run on the test function f_8, and restarted until at least 30 seconds had passed (according to Figure 2 in Hansen et al. [50]). These experiments have been conducted with an Intel Core 2 Duo 2.00 GHz processor computer under Windows XP using the MATLAB 7.6.0.324 version. We have completed two experiments using the BFGS and the simplex local search methods. The other algorithm parameters were the same as before. In the first case (BFGS) the results were $(2.8, 2.9, 3.0, 3.0, 3.2, 3.2) \cdot 10^{-4}$ seconds, while in the second case (Nelder-Mead simplex) they were $(2.6, 2.9, 3.4, 4.6, 7.5,\ 21.0) \cdot 10^{-4}$ seconds per function evaluation in dimensions 2, 3, 5, 10, 20, and 40, respectively. The CPU time of a function evaluation of the BFGS search grows sub-linearly with the dimension. For the Nelder-Mead simplex method, the CPU time increases with the dimension linearly up to 20 dimensional problems, while for 40 dimensional functions a rapid increase can be observed.

2.4.3.4 Results and discussion

The GLOBAL method have been tested in a black-box scenario on 24 noiseless benchmark functions. Results from experiments according to Hansen et al. [50] on the benchmark functions given in Finck et al. [41], and Hansen et al. [53] are presented in the Figure 2.3 and Tables 2.7 and 2.8.

For low search space dimensions the algorithm shows good results on many functions. The number of solved functions amounts to 18, 16, 11, 8, 5 out of 24 functions for dimensions 2, 3, 5, 10, 20. We can notice that GLOBAL obtains the highest number of successful trials in separable, moderate, illconditioned and weak structure noiseless functions, specifically for f_1, f_2, f_5, f_6, f_8, f_9, f_{10}, f_{11}, f_{12}, f_{21} and f_{22} in dimensions 2, 3, and 5. For f_1, f_2, f_5, f_8 and f_9, the method obtained successful trials for all dimensions (see Figure 2.3 and Tables 2.7 and 2.8).

The scaling of the expected number of function evaluations with the problem dimension is closely linear for f_8, f_9, is ca. quadratic for f_2. For f_1 and f_5 we can observe a decreasing tendency (see Figure 2.3). These results are due to the stochastic nature of the method, usually the ERT grows sub-linearly on these functions. The running times to reach the final target function value in case of the solved problems in 20 dimension range between $25n$ and $2,500n$.

Considering the different function subgroups, the best behavior of the GLOBAL algorithm can be observed on the separable (except f_3 and f_4), moderate (except f_7) and ill-conditioned functions. The good performance on these subgroups are due to the unimodal property of the objective functions. On the other hand, most of these functions have a quite large region of attraction of the global optimum (i.e. f_8 and f_9), hence there is a high chance to sample in this ones. In case of the f_6 (Attractive sector) and f_{11} (Discus) functions in dimension 20 up to the target precision 10^{-1}, all the 15 trials were successful, but the method fails to reach $\Delta f = 10^{-3}$. The results on the attractive sector function can be improved by increasing the function evaluation limit of the BFGS method, while for the Discuss function f_{11} one cannot find better target precision value than 10^{-1} in 20-D due to a problem of the BFGS local search. In this case the local search method stops too early because it cannot decrease the objective function along the current search direction. Finding the final target function values for f_8 and f_9 are mainly due to the BFGS local search and partly to the property of these functions presented previously. GLOBAL performs also well on the Gallagher's multimodal functions f_{21} and f_{22} with weak global structure. Compared to the best algorithm from BBOB-2009, the GLOBAL method can improve the ERT in dimensions 5 and 20 on the latter functions.

The hardest problems for which the method didn't reach the solution in higher dimensions are the multimodal Rastrigin functions f_3, f_4, f_{15}, and f_{24}. In case of the last one even in 2-D we cannot find a better target precision value than 10^{-2}, while in the case of f_3, f_4, f_{15} functions the Δf_{best} value is not better than 10 in 5-D and 10^2 in 20-D, respectively. The common feature of these functions is that they have more than 10^n local optima. Therefore the algorithm cannot discover the overall function structure. Moreover, the size of the region attraction of the global optimum is small for this problems, and hence the algorithm fails to satisfactorily sample in these regions. GLOBAL also fails to reach a target value below 1 on the f_{17} and f_{19} multimodal functions with adequate global structure in 5-D and 10 in 20-D. The reason is the same as presented above.

Considering the individual maximum number of function evaluations, GLOBAL performs well on ill-conditioned functions and on the multimodal weakly structured functions for a budget smaller than a thousand times n.

Summarizing our results obtained, the GLOBAL algorithm performs well especially on functions with moderate number of local minima using a small budget of function evaluations. Similar conclusion has been reached in Hansen et al. [52], where 31 algorithms were compared on a testbed of functions in dimensions up to 40. GLOBAL was ranked together with NEWUOA (see Powell [93]) and MCS (see Huyer and Neumaier [59]) best for a function evaluation budget of up to $500n$ function values, but was no longer competitive when the budget was significantly larger.

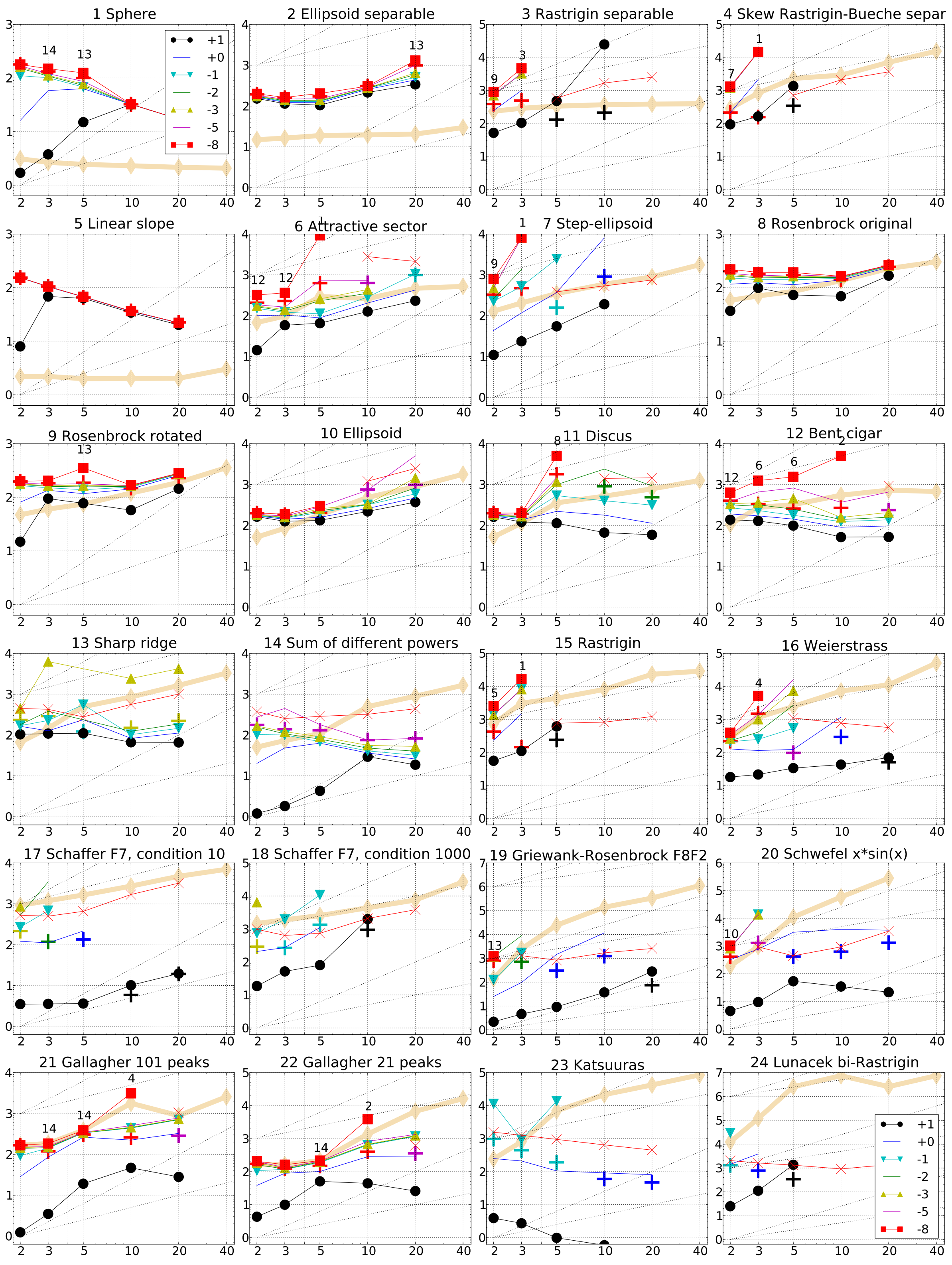


Figure 2.3: Expected running time (ERT) divided by dimension versus dimension in log-log presentation. Shown are different target values $f_{\text{opt}} + \Delta f$, where $\Delta f = 10^{\{+1,0,-1,-2,-3,-5,-8\}}$ and the exponent is given in the legend of f_1 and f_{24}. Plus symbols (+) show the median number of f-evaluations for the best reached target value. Crosses (×) indicate the total number of f-evaluations (#FEs($-\infty$)) divided by the number of trials. Numbers above ERT-symbols indicate the number of successful trials. Y-axis annotations are decimal logarithms.

Δf	1e+1	1e+0	1e-1	1e-3	1e-5	1e-7	#succ
f₁	11	12	12	12	12	12	15/15
	6.8(9.4)	26(0.61)	28(0.74)	32(1.2)	35(1.0)	39(1.4)	13/15
f₂	83	87	88	90	92	94	15/15
	6.3(1.9)	6.9(1.9)	7.3(1.8)	7.8(1.5)	8.2(1.4)	8.5(1.4)	15/15
f₃	716	1,622	1,637	1,646	1,650	1,654	15/15
	3.3(3.7)	∞	∞	∞	∞	∞*2,600*	0/15
f₄	809	1,633	1,688	1,817	1,886	1,903	15/15
	8.3(9.2)	∞	∞	∞	∞	∞*3,200*	0/15
f₅	10	10	10	10	10	10	15/15
	32(1.3)	33(2.6)	34(2.4)	34(2.4)	34(2.4)	34(2.4)	15/15
f₆	114	214	281	580	1,038	1,332	15/15
	2.9(0.21)	2.1(0.60)	2.0(0.50)	2.2(1.6)	3.6(3.9)	35(37)	1/15
f₇	24	324	1,171	1,572	1,572	1,597	15/15
	12(6.7)	5.7(5.8)	10(11)	∞	∞	∞*1,900*	0/15
f₈	73	273	336	391	410	422	15/15
	5.0(0.34)	2.1(1.3)	2.1(1.1)	2.1(0.86)	2.1(0.86)	2.2(0.81)	15/15
f₉	35	127	214	300	335	369	15/15
	11(2.2)	4.6(1.3)	3.2(0.80)	2.8(0.74)	2.7(0.60)	2.7(1.2)	13/15
f₁₀	349	500	574	626	829	880	15/15
	1.9(0.70)	1.6(0.49)	1.8(0.71)	2.0(1.5)	1.7(1.1)	1.7(1.1)	15/15
f₁₁	143	202	763	1,177	1,467	1,673	15/15
	4.0(1.5)	5.5(2.6)	3.5(3.2)	5.0(6.2)	5.0(6.5)	8.5(8.3)	8/15
f₁₂	108	268	371	461	1,303	1,494	15/15
	4.6(1.2)	2.7(0.61)	2.4(0.82)	5.0(6.0)	3.1(4.0)	3.4(3.9)	6/15
f₁₃	132	195	250	1,310	1,752	2,255	15/15
	4.2(2.5)	6.1(4.9)	11(11)	∞	∞	∞*1,300*	0/15
f₁₄	10	41	58	139	251	476	15/15
	2.2(1.9)	7.7(0.22)	5.9(0.28)	3.3(0.40)	3.6(2.1)	∞*1,300*	0/15
f₁₅	511	9,310	19,369	20,073	20,769	21,359	14/15
	6.0(6.9)	∞	∞	∞	∞	∞*2,700*	0/15
f₁₆	120	612	2,663	10,449	11,644	12,095	15/15
	1.4(1.3)	1(0.53)	1(1.2)	3.5(4.2)	6.8(8.0)	6.6(7.7)	0/15
f₁₇	5.2	215	899	3,669	6,351	7,934	15/15
	3.5(3.1)	5.0(4.0)	∞	∞	∞	∞*3,100*	0/15
f₁₈	103	378	3,968	9,280	10,905	12,469	15/15
	3.9(1.7)	15(15)	14(14)	∞	∞	∞*2,600*	0/15
f₁₉	1	1	242	1.20e5	1.21e5	1.22e5	15/15
	46(44)	7,329(8,077)	∞	∞	∞	∞*4,300*	0/15
f₂₀	16	851	38,111	54,470	54,861	55,313	14/15
	17(4.9)	18(19)	∞	∞	∞	∞*2,300*	0/15
f₂₁	41	1,157	1,674	1,705	1,729	1,757	14/15
	2.3(2.2)	1.1(0.87)	1(0.85)	1(0.83)	1(0.82)	1(0.81)	14/15
f₂₂	71	386	938	1,008	1,040	1,068	14/15
	3.6(1.7)	1.3(0.90)	1(1.1)	1(1.1)	1(1.1)	1(1.0)	14/15
f₂₃	3.0	518	14,249	31,654	33,030	34,256	15/15
	1.6(2.0)	1.0(0.48)	4.8(5.6)	∞	∞	∞*4,900*	0/15
f₂₄	1,622	2.16e5	6.36e6	9.62e6	1.28e7	1.28e7	3/15
	4.2(4.7)	∞	∞	∞	∞	∞*6,400*	0/15

Table 2.7: Expected running time (ERT) and half-interquantile range (90% – 10%) in number of function evaluations divided by the best ERT measured during BBOB 2009 and 2010 (given in the respective first row) for different Δf values for functions f_1–f_{24} in 5-D.

Δf	1e+1	1e+0	1e-1	1e-3	1e-5	1e-7	#succ
f₁	43	43	43	43	43	43	15/15
	8.0	8.0	8.0	8.0	8.0	8.0	15/15
f₂	385	386	387	390	391	393	15/15
	18(3.7)	23(3.0)	26(13)	33(14)	51(40)	63(65)	13/15
f₃	5,066	7,626	7,635	7,643	7,646	7,651	15/15
	∞	∞	∞	∞	∞	∞*5.0e4*	0/15
f₄	4,722	7,628	7,666	7,700	7,758	1.41e5	9/15
	∞	∞	∞	∞	∞	∞*7.8e4*	0/15
f₅	41	41	41	41	41	41	15/15
	10(0.52)	11(0.78)	11(0.78)	11(0.78)	11(0.78)	11(0.78)	15/15
f₆	1,296	2,343	3,413	5,220	6,728	8,409	15/15
	3.6(1.00)	3.6(0.74)	6.1(3.0)	∞	∞	∞*4.1e4*	0/15
f₇	1,351	4,274	9,503	16,524	16,524	16,969	15/15
	∞	∞	∞	∞	∞	∞*1.4e4*	0/15
f₈	2,039	3,871	4,040	4,219	4,371	4,484	15/15
	1.6(0.32)	1.2(0.16)	1.2(0.16)	1.2(0.15)	1.2(0.14)	1.2(0.14)	15/15
f₉	1,716	3,102	3,277	3,455	3,594	3,727	15/15
	1.7(0.28)	1.7(0.89)	1.6(0.84)	1.6(0.79)	1.6(0.77)	1.5(0.74)	15/15
f₁₀	7,413	8,661	10,735	14,920	17,073	17,476	15/15
	1(0.22)	1.1(0.15)	1.1(0.53)	2.0(1.7)	5.9(6.8)	∞*4.1e4*	0/15
f₁₁	1,002	2,228	6,278	9,762	12,285	14,831	15/15
	1.2(0.42)	1.0(0.60)	1(0.84)	∞	∞	∞*2.7e4*	0/15
f₁₂	1,042	1,938	2,740	4,140	12,407	13,827	15/15
	1(0.85)	1(0.88)	1(0.70)	1(0.49)	1.1(1.1)	3.4(3.4)	0/15
f₁₃	652	2,021	2,751	18,749	24,455	30,201	15/15
	2.0(0.34)	1.1(0.08)	1.1(0.04)	4.5(5.4)	∞	∞*1.9e4*	0/15
f₁₄	75	239	304	932	1,648	15,661	15/15
	5.0(0.28)	2.2(0.22)	2.1(0.21)	1.1(0.08)	1(0.04)	∞*8,500*	0/15
f₁₅	30,378	1.47e5	3.12e5	3.20e5	4.49e5	4.59e5	15/15
	∞	∞	∞	∞	∞	∞*2.4e4*	0/15
f₁₆	1,384	27,265	77,015	1.88e5	1.98e5	2.20e5	15/15
	1(0.72)	∞	∞	∞	∞	∞*1.2e4*	0/15
f₁₇	63	1,030	4,005	30,677	56,288	80,472	15/15
	6.2(1.2)	∞	∞	∞	∞	∞*6.9e4*	0/15
f₁₈	621	3,972	19,561	67,569	1.31e5	1.47e5	15/15
	∞	∞	∞	∞	∞	∞*7.5e4*	0/15
f₁₉	1	1	3.43e5	6.22e6	6.69e6	6.74e6	15/15
	5,601(3,531)	∞	∞	∞	∞	∞*5.3e4*	0/15
f₂₀	82	46,150	3.10e6	5.54e6	5.59e6	5.64e6	14/15
	5.2(0.38)	1.6(1.7)	∞	∞	∞	∞*7.8e4*	0/15
f₂₁	561	6,541	14,103	14,643	15,567	17,589	15/15
	1(0.26)	1(1.3)	1(1.2)	1(1.1)	1(1.1)	2.1(2.3)	0/15
f₂₂	467	5,580	23,491	24,948	26,847	1.35e5	12/15
	1.1(0.54)	1(1.5)	1(1.1)	1(1.1)	1(0.95)	1.3(1.5)	0/15
f₂₃	3.2	1,614	67,457	4.89e5	8.11e5	8.38e5	15/15
	2.8(2.7)	1(0.93)	∞	∞	∞	∞*9,300*	0/15
f₂₄	1.34e6	7.48e6	5.19e7	5.20e7	5.20e7	5.20e7	3/15
	∞	∞	∞	∞	∞	∞*2.8e4*	0/15

Table 2.8: Expected running time (ERT) and half-interquantile range (90% – 10%) in number of function evaluations divided by the best ERT measured during BBOB 2009 and 2010 (given in the respective first row) for different Δf values for functions f_1–f_{24} in 20-D.

2.5 Application: The flexible retirement problem

2.5.1 The model

We analyze the problem of designing a stable pension scheme (see Eső and Simonovits [38, 39]). There is a (stationary) population of individuals who have private information regarding their life expectancies (denoted by an integer t, calculated from the start of their careers). Every individual enters the labor market at the professional age 0, and produces 1 unit of goods per year while she or he is active, 0 when the employee is inactive (e.g. when retired or dead).

The pension systems we consider will be realistic in the following aspects. The first ingredient of a pension scheme is a yearly social security contribution rate, $\tau < 1$, which is levied on active workers. When an employee retires after R active years, she or he stops producing goods and paying the contribution, and receives a yearly retirement benefit of $b > 0$, until the end of her or his life. The government designs the contribution rate τ, and the benefit schedule as a function of the year of retirement, $b(R)$. We require that the pension system be financially sound, that is the benefit payments cannot exceed the amount of social security contributions paid.

An individual's lifetime utility, v, is the sum of his or her total income during the active and retired periods. If a worker of type t retires at age R, then she or he receives utility or felicity $u(1-\tau)$ for R years and $w(b)$ for $(t-R)$ years, and the lifetime utility is then

$$v = Ru(1-\tau) + (t-R)w(b).$$

The individual's preference for leisure is reflected in that $u(\cdot)$ and $w(\cdot)$ are different functions. For simplicity, we may assume that $u(x) = w(x) - \epsilon$, and $\epsilon > 0$, where ϵ is the constant disutility of labor.

We consider a discrete-type model. Types of workers (the life expectancies) range from S to T (both integers). To avoid triviality, we assume that there are at least two different types, i.e. $S < T$. Let f_t be the relative frequency of individuals with a life expectancy of t: $f_S, f_T > 0$ and $\sum_{t=S}^{T} f_t = 1$.

An individual's *balance* is the difference between the expected lifetime contributions and expected lifetime benefits:

$$z = \tau R - b \cdot (t-R) = (\tau + b)R - tb.$$

The government's goal is to design an optimal pension system, described by $(b(R), \tau)$, maximizing an additive concave social welfare function. We can split the government's problem into two subproblems, first considering the optimal choice of $b(R)$ for a fixed τ, then we can optimize over τ given the solution to the optimal $b(R)$ schedule for all τ-s. In the analysis below we will focus on the first issue, on the determination of $b(R)$ for a given τ.

Since τ is given for now, we denote $u(1-\tau)$ by $\bar{u}$. Denote further the lifetime utility of a worker with life expectancy t by v_t, where

$$v_t = [\bar{u} - w(b_t)]R_t + w(b_t)t.$$

We assume that individuals have private information (better just an estimation) regarding their life expectancies, and only the distribution of these data is commonly known. Therefore the optimal benefit retirement schedule will have to satisfy all incentive compatibility constraints. The incentive compatibility of $(b_t, R_t)_{t=S}^T$ means that type t prefers to choose (b_t, R_t) from the schedule. The constraints are the following:

$$v_t + w(b_t) \leq v_{t+1} \leq v_t + w(b_{t+1}), \text{ for } t = S, \dots T-1.$$

For a given τ, the social planner maximizes the frequency-weighted sum of an increasing and concave function ψ of the individual utilities under constraints. The optimization problem form of it is:

$$\max_{(b_t,R_t)_t} \sum_{t=S}^{T} \psi(v_t) f_t, \tag{2.9}$$

subject to

$$v_t = [\bar{u} - w(b_t)]R_t + w(b_t)t, \quad t = S, \dots T, \tag{2.10}$$

$$\sum_{t=S}^{T} [(\tau + b_t)R_t - tb_t] f_t = 0, \tag{2.11}$$

$$v_{t+1} = v_t + w(b_t), \ t = S, \dots, T-1. \tag{2.12}$$

It is easy to observe that the model is a nonlinear constrained optimization problem with $2(T - S + 1)$ variables to be optimized. On the basis of the problem structure, we can simplify it to have a smaller dimensional problem with less constraints to meet. Let us thus consider $(b_t, v_t)_t$ as variables instead of $(b_t, R_t)_t$. In this case for known b_t, $t = S, \dots, T$ and v_S the v_t, $t = S+1, \dots, T$ can be determined from the (2.12) equations, thus we can reduce the problem to a system of $T - S + 2$ unknown variables.

In order to solve the problem we used the GLOBAL algorithm for the negative of the objective function (2.9). The bounds for the variables b_t, $t = S, \dots, T$, and v_S were chosen in such a way that they correspond to the conditions of the problem. To be able to obtain reasonable results, we have added two new constraints. The first required that the resulting sequence of b_t and v_t values are increasing, the second forced that the last b value cannot be larger than 0.8. We composed the penalty functions corresponding to the (2.10) and (2.11) constraints and they were added to the objective function.

2.5.2 Computational test

For testing purposes we consider the next functions and constants, and in this way we fix the value and the definition of them:

- Let the pensioner's felicity function be $w(x) = \theta + x^\sigma/\sigma$, $\sigma < 1$.
- The social welfare function is set to $\psi(v) = v^\rho/\rho$, for a $\rho \leq 1$, $\theta = 4.1$.
- The uniformly distributed life expectancies are $f_t \equiv 1/(T - S + 1)$.

Table 2.9: GLOBAL test result, the optimal retirement model calculated back for the original decision variables.

t	b_t	R_t	z_t	v_t
49	0.7315747141	42.4540	3.7019300	31.3126
50	0.7378840212	42.5119	2.9770440	33.0743
51	0.7420314598	42.5541	2.2436582	34.8461
52	0.7454805416	42.5926	1.5054927	36.6243
53	0.7457289677	42.5957	0.7603081	38.4079
54	0.7537686858	42.7018	0.0241403	40.1919
55	0.7558273086	42.7308	-0.7272744	41.9883
56	0.7569907426	42.7483	-1.4817221	43.7878
57	0.7860244936	43.1915	-2.2155473	45.5891
58	0.7886071091	43.2311	-3.0006520	47.4332
59	0.8000000000	43.4126	-3.7873769	49.2811

– The individuals' life expectancies are between $S = 49$ and $T = 59$ (after the career start).

– The contribution rate is $\tau = 0.2$, and we set $\bar{u} = 0.466$, $\sigma = 0.2$, $\rho = -1$.

These are to a certain extent realistic settings, and still they allow to have a relatively simple constrained nonlinear optimization problem. Since we have not utilized the special problem structure, for a larger dimensional problem with measured frequency values etc., we could expect to have similar computational complexity and rate of success for GLOBAL.

The algorithm parameters of GLOBAL were set such that reflect the difficulties of the problem. The sample size was given as 20,000. This value is relatively high, and together with the number of best points kept after transforming the initial sample, 15 it ensures a high level of reliability – at the cost of larger number of objective function evaluations. The precision of the local search procedure was required to be at least 8 decimal digits. This value is well over what we really need in the minimizer points and much beyond the precision of the data used in the problem description. Still it is needed to have a good reliability, since the recognition of the regions of attractions around local minimizer points is critical. The local search method selected was UNIRANDI, since the penalty function approach caused our objective function to became not differentiable on several places.

The best objective function value found by GLOBAL was -0.0253684. It is close to and a bit better than the earlier know value. The corresponding (b_t, R_t) and z_t, v_t values are listed in Table 2.9. The necessary CPU time was closely half a minute to one minute per runs. When repeating the same numerical test we obtained similar results, which is an indication of the reliability of the GLOBAL algorithm.

With the same algorithm parameters we have run the program ten times independently. The obtained efficiency and precision results are comprised in Table 2.10. The first column contains the serial number of the numerical test run, the second has the number of function evaluations that was necessary to reach that result, in the third the necessary CPU time is given in seconds. The fourth column contains the summarized

Table 2.10: The results of ten independent runs of GLOBAL on the investigated pension system design problem: the serial number of the run, the number of function evaluations and CPU time needed (in seconds), then the absolute difference to meet the constraints and the obtained approximative global maximum value.

#	NFE	CPU time	constraint difference	approx. optimum
1.	105,188	42	0.000000011	-0.0253720
2.	112,574	45	0.000000065	-0.0253740
3.	70,647	31	0.000000064	-0.0253684
4.	77,484	30	0.000000041	-0.0253713
5.	100,912	40	0.000000003	-0.0253838
6.	125,188	50	0.000000293	-0.0253835
7.	95,150	37	0.000000007	-0.0253686
8.	97,521	38	0.000000076	-0.0253689
9.	110,521	43	0.000000022	-0.0253700
10.	200,752	79	0.000000034	-0.0253704

absolute residuals of the constraints on the approximate optimum point, while the last column gives the best objective function reached during that run.

The conclusion of the numerical test is that on the investigated real-life global optimization problem, the improved GLOBAL algorithm was able to find good approximations of the global minimizer points while the amount of computational efforts needed remained limited and in the acceptable region. The precision of the estimated global maximum value and the absolute difference between the two sides of the constraint equations can be improved further at the cost of higher CPU time and number of function evaluations. However the present figures are already acceptable.

Chapter 3

Interval global optimization methods

3.1 Introduction

There are several types of errors that are generated in numerical computations using floating point arithmetic. These errors may be rounding errors due to finite representation of numbers, or errors due to uncertain values of parameters in mathematical models. Let us consider the famous example (see Rump [101]) which demonstrates the effect of rounding error. The Rump's expression is:

$$f(x,y) = (333.75 - x^2)y^6 + x^2(11x^2y^2 - 121y^4 - 2) + 5.5y^8 + \frac{x}{2y}. \qquad (3.1)$$

The evaluation of (3.1) in $x_0 = 77617$ and $y_0 = 33096$, using the round-to-nearest IEEE-754 arithmetic produces:

$$\begin{aligned} 32 - \text{bit} : f(x_0, y_0) &= 1.172604 \\ 64 - \text{bit} : f(x_0, y_0) &= 1.1726039400532 \\ 128 - \text{bit} : f(x_0, y_0) &= 1.1726039400531786318588349045201838 \end{aligned}$$

All three results agree in the first seven decimal digits. Nevertheless, they are all completely incorrect. Even their sign is incorrect. The correct result is:

$$f(x_0, y_0) = -0.827396059946821368141165095479816...$$

Interval computation provides a tool for estimating and controlling these errors by providing upper and lower bounds on them, and thus developing numerical methods that yield reliable results. A major focus on interval analysis is to develop practical interval algorithms that produce bounds as narrow as possible on the solution of numerical computing problems. The concept of interval analysis is to compute with intervals of real numbers in place of real numbers.

The idea of bounding rounding errors using intervals was introduced by several researchers in the 1950's. Interval analysis was first introduced in Moore [74]. Since then, many of articles have appeared on the subject.

Interval algorithms may be used in most areas of numerical analysis, and are used in many applications such as engineering problems, robotics, finance and decision making, and computer aided design. Another important application is in computer-assisted proofs. Several conjectures have recently been proven using interval analysis, perhaps the most famous of them is Kepler's conjecture[1], which remained unsolved for nearly 400 years.

This chapter deals with interval based global optimization algorithms and it is based on three papers: Csendes and Pál [32], Pál and Csendes [83], and Pál and Csendes [85].

In the first part of the chapter, we review the basic concepts of interval computation. After that, we describe the most important interval techniques used by interval branch and bound methods.

The following subsections deals with the presentation of a basic and an advanced algorithm for the bound constrained global optimization problem implemented in MATLAB using the INTLAB package. The performance of the algorithms is studied through extensive numerical experiments on some well known functions by comparing with the similar C-XSC based method. The contribution of the present author was the implementation of interval global optimization algorithms in MATLAB and the computation of the numerical tests. The discussion part of the numerical results was a common work with the coauthor.

We also describe and test an algorithm using a new condition for applying the Newton step. The theoretical aspects of it are also investigated. The application of a new condition was suggested by the coauthor, while the numerical tests were done by the present author. The discussion part of the numerical results was a common work with the coauthor. The theoretical part was also done by the present author.

The last section deals with an application from the domain of sensor networks, namely the localization problem of sensor networks. The numerical tests and their discussion was completed by the present author. It is worth mentioning that we also worked on an another application of interval arithmetic, namely on the efficient estimation of loads in service networks (see Pál and Csendes [85]). In the application we use intervals to represent uncertainty data and interval arithmetic operations for calculations. The obtained results are not included in the present thesis.

3.2 Interval computation

3.2.1 Interval arithmetic

A *real interval* X is the bounded, closed subset of the real numbers defined by

$$X = [\underline{X}, \overline{X}] = \{x \in \mathbb{R} \mid \underline{X} \le x \le \overline{X}\},$$

where $\underline{X}, \overline{X} \in \mathbb{R}$ and $\underline{X} \le \overline{X}$. $\underline{X}$ is called the lower bound, while $\overline{X}$ is the upper bound.

An n-dimensional *interval vector* or *box* $X = (X_1, X_2, \ldots, X_n)$ is defined to be a vector with interval components $X_i,\ i = 1, \ldots, n$.

The set of all intervals over $\mathbb{R}$ is denoted by $\mathbb{I} := \{[a, b] \mid a \le b, a, b \in \mathbb{R}\}$ and the set of n-dimensional interval vectors by $\mathbb{I}^n$. $\mathbb{I}(X)$ stands for all intervals in X.

[1]http://en.wikipedia.org/wiki/Kepler_conjecture

The *width* of the one-dimensional X interval is defined by

$$\mathrm{w}(X) = \overline{X} - \underline{X},$$

while the width of an n-dimensional interval vector is

$$\mathrm{w}(X) = \max_{i=1,\ldots,n} (\overline{X}_i - \underline{X}_i).$$

The *midpoint* of the interval X is defined by $m(X) = (\underline{X} + \overline{X})/2$, if $X \in \mathbb{I}$, and $m(X) = (m(X_1), m(X_2), \ldots, m(X_n))$, if $X \in \mathbb{I}^n$.

In interval computation we use intervals instead of real numbers. Introduced in its modern form by R.E. Moore (see Moore [74]), real interval arithmetic is based on an arithmetic within the set of closed intervals of real numbers.

Definition 3.1. *Let "$\circ$" denote one of the four elementary arithmetic operators* $\{+, \text{-}, *, /\}$. *For* $A, B \in \mathbb{I}$ *we define*

$$A \circ B = \{a \circ b \mid a \in A \;\text{ and }\; b \in B\}, \tag{3.2}$$

where A/B *is defined only if* $0 \notin B$.

The four arithmetic operations are continuous mappings of $\mathbb{R}^2$ onto $\mathbb{R}$ and A, B are bounded and closed. Therefore, also $A \circ B$ is an interval. The definition ensures that the resulting interval contains all possible outcomes from applying "$\circ$" with operands from A and B. The operations on intervals may also be calculated explicitly as

$$\begin{aligned}
[a,b] + [c,d] &= [a+c, b+d], \\
[a,b] - [c,d] &= [a-d, b-c], \\
[a,b] * \ [c,d] &= [\min\{ac, ad, bc, bd\}, \max\{ac, ad, bc, bd\}], \\
[a,b] \ / \ \ [c,d] &= [a,b] * [1/d, 1/c], \text{if } 0 \notin [c,d].
\end{aligned}$$

The most important properties of the interval arithmetic operations are the following:

– The addition and multiplication operations are commutative and associative.

– The distributive law doesn't hold, instead there is a sub-distributive law:

$$A \cdot (B + C) \subset A \cdot B + B \cdot C, \text{ where } A, B, C \in \mathbb{I}.$$

– The sub-distributive law is related to the *dependency problem.* This means that algebraic expressions that are equivalent in real arithmetic give different results when evaluated for intervals. For example, $X^2 - X$ over $X = [0,1]$ gives $[-1,1]$ and $X(X-1)$ gives $[-1,0]$ as a result.

– Subtraction and division in $\mathbb{I}$ are not the inverse operations of addition and multiplication respectively as in $\mathbb{R}$.

– Although there is an additive identity $[0,0]$ and a multiplicative identity $[1,1]$, additive and multiplicative inverses do not exist. The example indicating this fact is:

$$[1,2] - [1,2] = [-1,3].$$

For the elementary interval operations, division by an interval containing zero is not defined. It is often useful to remove this restriction to give what is called *extended interval arithmetic.* Extended interval arithmetic must satisfy (3.2), which leads to the following rules:

$$[a,b]/[c,d] = \begin{cases} [b/c,\infty)\,, & \text{if } b \le 0 \text{ and } d = 0, \\ (-\infty,b/d] \cup [b/c,\infty), & \text{if } b \le 0,\ c < 0, \text{ and } d > 0, \\ (-\infty,b/d]\,, & \text{if } b \le 0 \text{ and } c = 0, \\ (-\infty,a/c]\,, & \text{if } a \ge 0 \text{ and } d = 0, \\ (-\infty,a/c] \cup [a/d,\infty), & \text{if } a \ge 0,\ c < 0, \text{ and } d > 0, \\ [a/d,\infty)\,, & \text{if } a \ge 0 \text{ and } c = 0, \\ (-\infty,\infty)\,, & \text{if } a < 0 \text{ and } b > 0. \end{cases}$$

These formulas are not applicable to every problem, but they are appropriate for solving linear equations in connection with the interval Newton method.

The floating-point interval extensions of the standard functions {exp, tan, sin, cos,...} can be defined similarly. If f is a function from $\mathbb{R}$ to $\mathbb{R}$, then $f(X) = \{f(x) \mid x \in X\}$. Furthermore, if we use the monotonic property of a standard function, the evaluation will be more easier. For instance,

$$\arctan(X) = [\arctan(\underline{X}), \arctan(\overline{X})].$$

For non-monotonic functions the situation is more complicated. Without additional analysis, sharp bounds are rarely computed for this functions.

3.2.2 Machine interval arithmetic

Let us consider the interval $X = [\underline{X}, \overline{X}]$, which could be an input of a problem or the result of some calculations. It may not be representable on a machine if $\underline{X}$ and $\overline{X}$ are not machine numbers. This problem could be solved using the *machine interval arithmetic.*

If $\underline{X}$, $\overline{X}$ are not machine numbers, then they must be rounded in the following way: the lower bound of the interval is rounded down to the largest machine number less than the exact result and the upper bound is rounded up to the smallest machine number greater than the actual result. This process is called *outward rounding.*

Thus, the basic principle of interval arithmetic is retained in machine interval arithmetic, that is, the exact unknown result is contained in the corresponding calculated interval, and the rounding errors are under control.

The standard IEEE-754 (see Stevenson [109]) for binary floating point arithmetic prescribes four rounding modes: nearest, round down, round up and rounding to zero. The current version is IEEE 754-2008[2], which was published in August 2008, it includes nearly all of the original IEEE 754-1985. Thus, rounding modes suitable for interval arithmetic are available on a wide variety of machines. For the common programming languages, there are extensions in which rigorous outward rounded interval arithmetic

[2]http://ieeexplore.ieee.org/servlet/opac?punumber=4610933

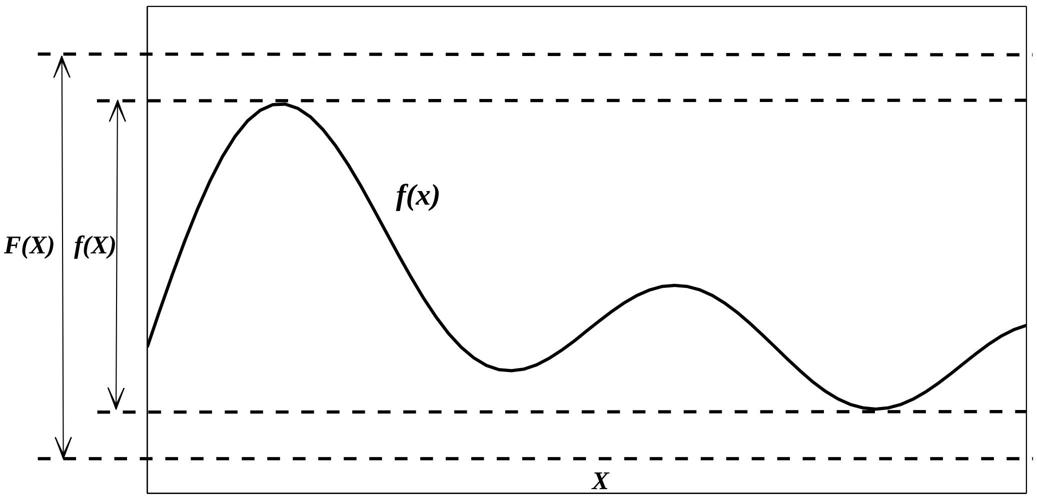


Figure 3.1: Inclusion function.

is easily available: INTLAB[3] (see Rump [102]) a toolbox for MATLAB, the Pascal extension PASCAL-XSC[4], the C extension C-XSC[5], and PROFIL/BIAS[6].

3.2.3 Inclusion functions

One of the basic problems of numerical computations is the calculation of the range of a function on an interval X with a computer. In general, it is impossible to compute the exact range or good safe bounds of it with floating point arithmetic. This problem can be solved with interval arithmetic. In the treatment of optimization problems using interval arithmetic, the main tool is the concept and application of *inclusion functions.*

Definition 3.2. *A function $F : \mathbb{I}^n \to \mathbb{I}$ is called an inclusion function of f in $X \subseteq \mathbb{R}^n$, when $x \in X$ implies $f(x) \in F(X)$. In other words,*

$$f(X) = \{f(x) \mid x \in X\} \subseteq F(X), \tag{3.3}$$

where $f(X)$ denotes the range of the function f on X (see Figure 3.1).

Inclusion functions for vector-valued or matrix-valued functions are defined analogously. The inclusion condition (3.3) must be satisfied in this case componentwise.

It turns out that interval analysis provides a natural framework for constructing inclusion functions recursively for a large class of functions.

Definition 3.3. *Consider a function $f : \mathbb{R}^n \to \mathbb{R}$, $(x_1, x_2, \ldots, x_n) \mapsto f(x_1, x_2, \ldots, x_n)$ expressed as a finite composition of the +, -, *, / operators and elementary functions. The natural interval extension of f is obtained by replacing each real variable x_i by an*

[3]http://www.ti3.tu-harburg.de/ rump/intlab/

[4]http://www.rz.uni-karlsruhe.de/ iam/html/language/pxsc.html

[5]http://www.xsc.de/

[6]http://www.ti3.tu-harburg.de/Software/PROFIL.html

interval variable and each operator or function by its interval counterpart. If each of the variables x_i, $i = 1, \dots, n$ occurs at most once in the expression of f, then the natural inclusion is minimal. In other words, the resulting interval enclosure will be the exact range $f(X)$.

Natural inclusion functions are not minimal in general, because of the dependency effect. The accuracy strongly depends on the expression of f, as illustrated by the following example.

Example 3.4. *Consider the following four expressions of the same function and evaluate their natural inclusion functions for $X = [-1, 1]$:*

$$\begin{aligned} f_1(X) &= X(X+1) = [-2, 2], \\ f_2(X) &= X * X + X = [-2, 2], \\ f_3(X) &= X^2 + X = [-1, 2], \\ f_4(X) &= (X + \frac{1}{2})^2 - \frac{1}{4} = [-\frac{1}{3}, 2]. \end{aligned}$$

An important property for inclusion functions is inclusion isotonicity, which is useful in convergence proofs.

Definition 3.5. *An inclusion function F is called inclusion isotone over $X \subseteq \mathbb{R}^n$, if $\forall\, Y, Z \in \mathbb{I}^n(X)$, and $Y \subseteq Z$ implies $F(Y) \subseteq F(Z)$.*

The natural inclusion function is isotone, since the four interval arithmetic operators and the standard elementary functions have this property.

A measure of the quality of an inclusion function F of f is the *excess-width*, $w(F(Y)) - w(f(Y))$, $Y \in \mathbb{I}^n(X)$. A measure for the asymptotic decrease of the excess-width as $w(Y)$ decreases is the *convergence order*.

Definition 3.6. *The convergence order of an inclusion function F of f over X is $\alpha > 0$, if the inequality*

$$w(F(Y)) - w(f(Y)) \leq C w^{\alpha}(Y),$$

holds for every $Y \in \mathbb{I}^n(X)$ for a real positive constant C.

Definition 3.7. *We say that the inclusion function F has the zero convergence property, if $w(F(Z_i)) \to 0$ holds for all the $\{Z_i\}$ interval sequences for which $Z_i \subseteq X$, for all $i = 1, 2, \dots, n$ and $w(Z_i) \to 0$.*

The natural inclusion function is α-convergent with $\alpha = 1$ (see Ratschek and Rokne [95]), and hence it has also the zero convergence property. In general, an inclusion function with larger convergence order is better, although it may also happen that an inclusion function with larger α has bigger overestimation for some intervals due to the constant C.

There are more sophisticated inclusion functions like the centered forms and slopes. Centered forms are inclusion functions with special features using also the gradient of the function. In this case we should compute the enclosure of the gradient. This is usually easy to obtain by automatic differentiation (see Griewank and Corlis [44]).

Many programming languages are now available that support interval datatypes and automatic differentiation (see the previous section).

The most important centered forms are the meanvalue form (see Moore [74]) and the Taylor form (see Berz and Hoffstätter [9], Neumaier [77]). The convergence order of the centered form is at least quadratic (see Krawczyk and Nickel [64]).

A primary use of interval derivative information is in bounding ranges or variations in functions over regions. However, it is not always necessary to use interval extensions of the derivative. One can obtain meanvalue forms with smaller widths if *slopes* (see Ratz [97]) instead of the inclusion of the first derivative are used.

3.3 Global optimization using interval arithmetic

Global optimization methods that use interval techniques provide rigorous guarantees that a global minimizer is found. Interval techniques are used to compute global information about functions over large regions. Most global optimization methods using interval techniques employ a branch and bound strategy. These algorithms decompose the search domain into a collection of boxes for which the lower bound on the objective function is calculated by an interval technique.

Just to name some of the numerous applications of global optimization using interval arithmetic, we point on some recent publications: these techniques were used to solve hard mathematical problems arising in the field of qualitative analysis of dynamical systems (see Csendes et al. [31, 30], and Bánhelyi et al. [6]) and discrete geometry, for optimal packing of circles in the square (see Markót and Csendes [67], and Szabó et al. [110]). Global optimization methods have also been applied for theoretical chemical problems in Balogh et al. [4], and for the evaluation of bounding methods in Tóth et al. [111].

Our aim was to provide an easy to use reliable global optimization method implemented in MATLAB. In this part a substantial work have been done on testing different algorithm variants. As a result, we found algorithms which reliability and efficiency are similar or better than the older ones.

3.3.1 Interval branch and bound methods

Consider the following specification of the general GOP:

$$\min_{x \in X} f(x), \tag{3.4}$$

where f is a twice continuously differentiable function defined over the n-dimensional interval $X \subset \mathbb{R}^n$. Our aim is to find the global minimizer points X^* of f and the corresponding minimum value f^*.

The basic idea of such interval branch-and-bound algorithms is to apply interval techniques to reject large regions in which the optimum can be guaranteed not to lie. For this reason, the original interval vector X gets subdivided (*branches*), and subregions which cannot contain a global minimizer of f are discarded (*bounding*). The other

Algorithm 9. The basic interval branch and bound framework

```
1.  function IntervalBranchAndBound($f, X$)
2.    $L_{res} := \emptyset$;  $L_{work} := \{X\}$
3.    while $L_{work} \neq \emptyset$ do
4.          Remove the first element ($X$) from the working list
5.          Calculate $F(X)$
6.          if $X$ cannot be discarded then
7.                Subdivide $X$ into $X^i$, $i = 1, \dots, p$ subintervals
8.                if $X^i$ fulfils some stopping criteria then
9.                      Enter $X^i$ in the result list
10.               else
11.                     Enter $X^i$ in the working list
12.               end
13.         end
14.   end
15.   return $L_{res}$
```

subregions get subdivided again until the desired accuracy (width) of the interval vectors is achieved.

The key concept of a bounding technique is the inclusion function and as we have seen in the previous section, interval arithmetic is able to provide these bounds in a number of ways such as natural extension, centered forms, and slopes.

The basic interval branch and bound framework steps are listed in Algorithm 9. The algorithm takes as input the objective function f and the initial box X.

In line 2, the result list (L_{res}) and the working list (L_{work}) are initialized. The algorithm contains a main iteration cycle and the steps from line 3 to line 14 will be repeated until the working list becomes empty. In line 4, we remove the first element of the working list while in line 5 the inclusion function will be computed over X. This last step is also called a bounding step. Interval arithmetic enables us to provide these bounds in a number of ways such as natural extension, centered forms, and slopes. In line 6, we try to discard the interval X using some *elimination rules.* If X cannot be rejected, we use *branching rules* (line 7) in order to get smaller intervals by subdividing X. In line 8, we check X^i whether it fulfils a proper *stopping criterion.* If so, then we put X^i in the result list (line 9), otherwise in the working list (line 11). We usually keep the lists ordered using different strategies. In line 15, the result list is returned.

In the subsequent sections we describe in more details the main steps of the algorithm.

3.3.1.1 Elimination rules

The main role of the elimination rules are discarding or shrinking certain elements from the working list that surely cannot contain a global minimizer point. These rules are also called *accelerating devices.*

The simplest and basic elimination rules are the *range check* and the *cut-off* test. We use the notation $\underline{f_X}$ as abbreviation for the lower interval bound of the interval function

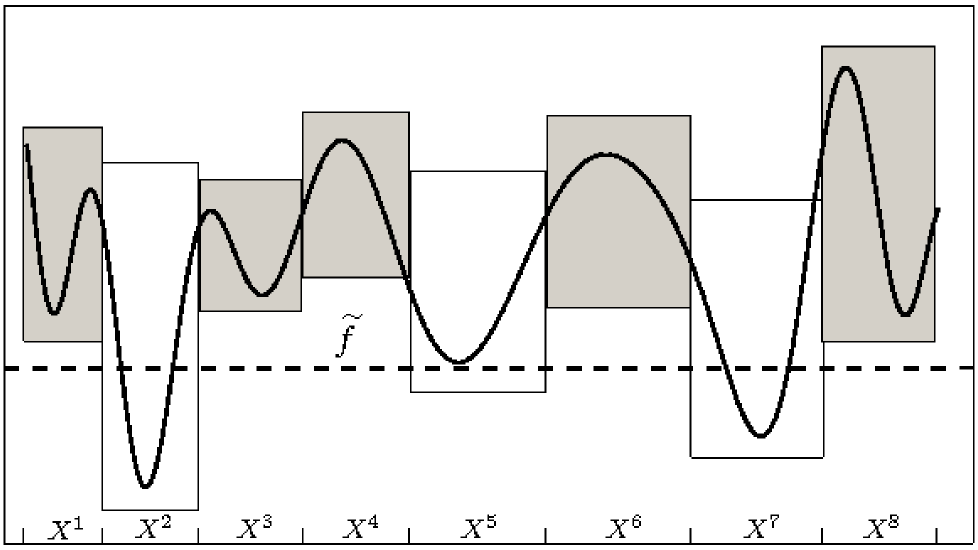


Figure 3.2: The cut-off test.

evaluation $F(X) := [\underline{f_X}, \overline{f_X}]$. Additionally, the algorithm uses a value $\widetilde{f}$ representing a guaranteed upper bound of the global minimum value, i.e. $\widetilde{f} \geq f^*$. The value $\widetilde{f}$ is calculated in the midpoint of the interval and it is updated during running the algorithm.

The value $\widetilde{f}$ is first used when newly subdivided intervals X^i are tested in a range check. If we know that $\underline{f_{X_i}} > \widetilde{f}$, then X^i cannot contain a global minimizer. The range check can be improved by incorporating centered forms.

In the cut-off test, by comparing the guaranteed upper bound $\widetilde{f}$ for the global minimum value f^* with the lower bound $\underline{f_X}$, we can discard all subintervals X in L_{work} for which

$$f^* \leq \widetilde{f} < \underline{f_X}.$$

Figure 3.2 illustrates the cut-off test, which deletes the intervals X^1, X^3, X^4, X^6 and X^8 in this special case.

In addition to the range check and cut-off test, the monotonicity test, the concavity test and the interval Newton method can be applied. The latter uses first and second order derivatives.

For a continuously differentiable function f, the *monotonicity test* determines whether the function is strictly monotone in an entire subinterval X. If f is strictly monotone in X, then X cannot contain a global minimizer in its interior. Therefore, if the interval enclosure $G := \nabla F(X)$ of ∇f evaluated over X satisfies

$$0 \notin G_i \text{ for some } i = 1, \ldots, n,$$

then the subinterval X can be deleted.

Figure 3.3 demonstrates the monotonicity test in the one-dimensional case for three subintervals of X. In this special case, X^1 can be reduced to the boundary point $\underline{X}^1$, X^2 remains unchanged, and X^3 can be deleted.

The *concavity test* discards intervals over which the objective function is strictly concave in a variable, since these intervals cannot contain a minimizer point inside.

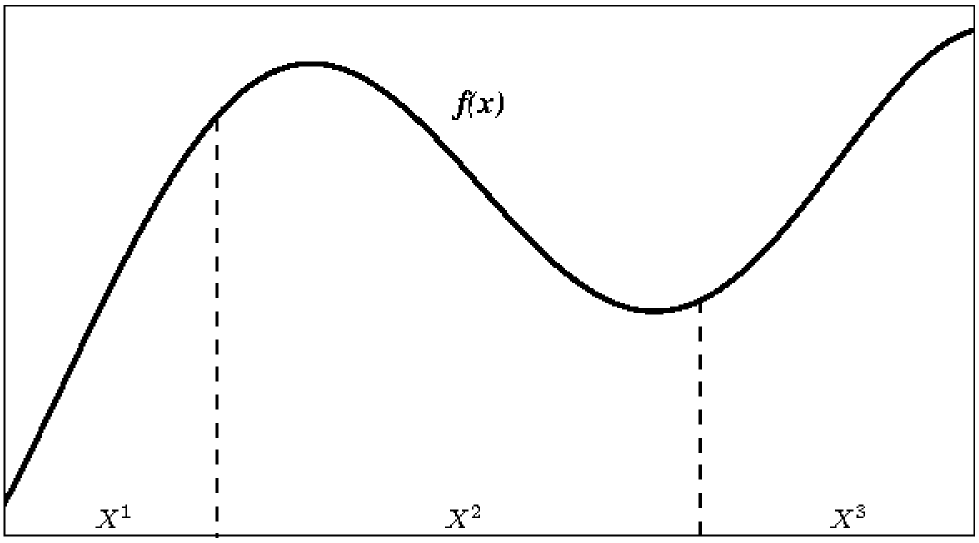


Figure 3.3: The monotonicity test.

We use an *interval Newton method* as part of the algorithm to solve the global optimization problem. In the method, we do not iterate the interval Newton algorithm to convergence. The reason for it is that it can be expensive, and it might be converging to a point that another procedure can show is not the global solution.

Therefore, we perform one pass of an interval Newton algorithm and then apply other procedures before doing another pass. In other words we apply one step of the extended interval Newton Gauss-Seidel method to the nonlinear system $\nabla f(x) = 0$ with $x \in X$, in order to improve the enclosure of X.

3.3.1.2 Branching rules

One of the most widely used branching rule is the simple bisection method (see Ratschek and Rokne [95]). In this case the actual interval is bisected through the longest edge into two equal size subintervals. We may also use multisection methods which split the current box through the longest edge into more than two equal size subintervals. According to some earlier studies (see Csallner et al. [26], Casado et al. [23]), multisection may improve the efficiency of interval branch and bound techniques. The direction selection of the splitting may also increase the convergence speed significantly.

We can apply different rules trying to find an optimal component (coordinate direction) to bisect the box X. Each of the rules selects a direction k by using a merit function:

$$k := \min\{j \mid j \in \mathcal{I} \quad \text{and} \quad D(j) = \max_{i=1}^{n} D(i)\},$$

where $\mathcal{I} = \{1, \ldots, n\}$ and $D(i)$ is determined by the following rules:

Rule A

This rule is the interval-width oriented rule (see Moore [74], and Ratschek and Rokne [95]), where the box is bisected through the longest edge. The rule chooses the coordinate direction with

$$D(i) := w(X_i), \tag{3.5}$$

where X_i is the i-th component of the interval X.

It is justified by the idea that if the original interval is subdivided in a uniform way as in this case, then the width of the actual subintervals goes to zero the most rapidly.

Rule B

The following function was proposed by G.W. Walster and presented in Hansen [49]:

$$D(i) := w(\nabla F_i(X))w(X_i), \tag{3.6}$$

where $\nabla F_i(X)$ is the i-th component of the inclusion of gradient of f. The rule approximates the change in f resulting from the i-th variable changing over X_i.

Rule C

This rule was defined by Ratz in Ratz [96], and the main idea was to minimize the width of the inclusion using the function:

$$D(i) := w(\nabla F_i(X)^T(X_i - m(X_i))). \tag{3.7}$$

The Rules B and C have the same merit function value if and only if either $\min \nabla F_i(X) = 0$ or $\max \nabla F_i(X) = 0$.

Rule D

This rule is derivative-free like Rule A and reflects the machine representation of the inclusion function $F(X)$ (see Hammer et al. [46]). It is defined by

$$D(i) := \begin{cases} w(X_i) & \text{if } 0 \in X_i, \\ w(X_i)/\min\{|x_i| \mid x_i \in X_i\} & \text{otherwise.} \end{cases} \tag{3.8}$$

This rule may decrease the excess width $w(F(X)) - w(f(X))$ of the inclusion function that is caused in part by the floating point computer representation of real numbers.

3.3.1.3 Interval selection rules

As can be observed, in line 4 of the Algorithm 9 the first element of the list is removed for processing. The computational efficiency of the algorithm depends strongly on which boxes we select from the list for subdivision. In the past, many selection strategies were developed which main aim was to maintain some order to work such that all of the boxes would be considered. Hence, no box was lost in the processing. The two most important selection rules were presented in Moore [74], Skelboe [107], and Hansen [47, 48].

The rule presented by R.E. Moore and S. Skelboe selects that box from the list which has minimum $\underline{f_X}$. It is based on the heuristic reflection that the smaller $\underline{f_X}$ is the larger the chances are for X to contain global minimizers.

Hansen's rule selects that box which is the oldest in L_{work}. Another rule selects the box which has maximum width (see Hansen [48]).

The above presented selection rules can be combined in the following way (see Ratz [96]): select the box which inclusion lower bound has the smallest value. If there are more than one box with this value, we select the oldest one. Experience has shown that the latter rule is preferable.

Traditionally, for subdivision, the algorithms selected a box with the smallest value of $\underline{f_X}$. There is an advanced subinterval selection criterion which is based on the RejectIndex indicator

$$pf^*(X) = \frac{f^* - \underline{F}(X)}{\overline{F}(X) - \underline{F}(X)},$$

where f^* is the known global minimum value.

This indicator was considered by many authors (see Casado and García [22], Casado et al. [23, 24]) and the studies have proven that this can provide reliable information on how close a subinterval in the search region is to a global minimizer point. The algorithm selects the next subinterval to be subdivided with the largest value of the indicator.

If the global minimum value is not available, we can use a wider class of indicators (see Csendes [28, 29]):

$$p(\widehat{f}, X) = \frac{\widehat{f} - \underline{F}(X)}{\overline{F}(X) - \underline{F}(X)},$$

where $\widehat{f}$ is an approximation of the global minimum.

3.3.1.4 Termination criteria

The choice of an appropriate termination criterion is essential in interval global optimization. Splitting and reducing boxes eventually causes any remaining box to be small. We usually require that one or more conditions be satisfied before a box is deemed to be small enough to be included in the set of solution boxes.

The termination criteria widely used can basically be arranged in three classes:

- A given accuracy for the inclusion of the optimum value.
- A sufficiently close inclusion for the set of global optimizers.
- A combination of the above two criteria.

If our primary goal is to calculate the optimum value of the objective function with a given accuracy, then the termination criterion has to reflect this aim. The simplest such stopping criterion is

$$w(f(X)) < \epsilon, \tag{3.9}$$

for a box X and for a given $\epsilon > 0$. Condition (3.9) guarantees that the global minimum value f^* of the objective function is bounded within the tolerance ϵ.

The second class of termination criteria consists of those concerning the accuracy of the inclusion of the global optimizers. The following two termination criteria were described in Hansen [47]. If X^* is at most denumerable and is to be included in a set with prescribed accuracy, then

$$\sum_{i=1}^{l_n} w(X_{ni}) < \epsilon, \tag{3.10}$$

or

$$w(X_{ni}) < \epsilon \text{ for } i = 1, \dots, l_n \tag{3.11}$$

will do, where l_n is the length and X_{ni} are the boxes of the current working list at the n-th iteration of the algorithm.

After having terminated, a list of boxes $X_{n1}, \dots, X_{nl_n}$ is left, and all we know is that

$$X^* \subseteq X_{n1} \cup \dots \cup X_{nl_n}.$$

We used the (3.9) criterion in our algorithms, in that sense that an X interval is added to the result list if the $w(f(X)) < \epsilon$ criterion is satisfied.

3.3.2 A basic algorithm

3.3.2.1 Description and implementation

The first method we present, is a simple algorithm for the bound constrained global optimization problem implemented in MATLAB using the INTLAB package. We choose MATLAB as a programming language, because it is a high-level technical computing language providing an interactive numerical environment for algorithm development, data visualization, and numerical computation. Furthermore, we should mention the popularity of MATLAB among the researchers and the integrability with external applications and languages, such as C, C++, Fortran, and Java.

INTLAB (INTerval LABoratory) is a toolbox for MATLAB supporting real and complex intervals. It uses the BLAS[7] routines which assure fast computing, comparable to pure floating point arithmetic. Beside the basic arithmetical operations, rigorous input and output, rigorous standard functions, gradients, slopes, automatic differentiation and multiple precision arithmetic is included.

The algorithm investigated now uses only a subroutine calculating the objective function as information on the global optimization problem, i.e. the expression is not required. The procedure does not apply the gradient and the Hessian of the objective

[7]http://en.wikipedia.org/wiki/Basic_Linear_Algebra_Subprograms

function, although these can be computed by the automatic differentiation facility of INTLAB. In other words, we study now that algorithm variant that does not assume the differentiability of the objective function.

This method originates in the Numerical Toolbox for Verified Computing (see Hammer et al. [46]), and it applies only a single accelerating device: the cutoff test – in contrast to the more sophisticated techniques studied in Pál and Csendes [83]. Now just natural interval extension (based on naive interval arithmetic) was applied to calculate the inclusion functions.

We use only simple bisection along the widest interval component, and no multisection and advanced subdivision direction selection (see Kearfott [62]). The subdivision direction is determined according to the well-tested and simple A subdivision direction selection rule (also used in Csendes [28]). The algorithm solves also one-dimensional problems.

For the MATLAB/INTLAB implementation we have followed closely the original C-XSC code which was developed for bound constrained global optimization by Mihály Csaba Markót based on the algorithm documented in Markót et al. [68]. The control structures of the two algorithms are basically the same, while the vectorial array statements of MATLAB were applied wherever possible.

The basic algorithm we have implemented is described by Algorithm 10. The algorithm takes as input the objective function f, the initial box X and a threshold value ϵ for the stopping criteria.

In line 2, the result list (L_{res}) and the working list (L_{work}) are initialized, while in line 3 an upper bound of the global minimum value $\widetilde{f}$ is computed. The main iteration starts (line 4) with an optimal direction selection procedure and with a simple bisection of the actual box, getting two new boxes. Between lines 7 and 19 we iterate over the bisected boxes $U^i, i = 1, 2$ and apply a range check (line 9) and a cut-off test (line 12). If the inclusion width of the actual interval U^i is less than ϵ, then it is stored in the result list (line 15), otherwise in the working list (line 17). The boxes are stored in the lists sorted in nondecreasing order with respect to the lower bounds of the inclusion function and in decreasing order with respect to the ages of the boxes (as a secondary sorting criterion).

The steps described previously are repeated until the working list becomes empty (line 21). When the main iteration stops, we compute a final enclosure (line 22) for the global minimum value and return the results (line 23).

3.3.2.2 Numerical tests and results

The numerical comparison aimed to clear whether the new implementation is capable to deliver similar quality results as the old one, and to measure the efficiency in terms of the usual indicators. Hence, we have completed a computational test, and compared the efficiency and the results of the INTLAB implementation to that of a C-XSC, BIAS, and Profil based procedure (see Markót et al. [68]).

For the test we used INTLAB version 5.4, MATLAB R2007a, and a PC with 1 Gbyte RAM and a 3 GHz Pentium-4 processor. The test problems included all the standard global optimization functions to be minimized, and basically all of those usually applied in comparing interval global optimization methods. Our tables contain results

Algorithm 10. The basic unconstrained interval global optimization algorithm

```
1.  function BasicGOP($f, X, \epsilon$)
2.    $L_{res} := \emptyset; L_{work} := \emptyset$
3.    $Y := X; \widetilde{f} := \overline{F}(m(X))$
4.    repeat
5.        OptimalComponent($Y, k_1$)
6.        Bisection($Y, k_1, U^1, U^2$)
7.        for $i = 1$ to 2 do
8.            $f_U := F(U^i)$
9.            if $\widetilde{f} < \underline{f_U}$ then continue
10.           if $\overline{F}(m(U^i)) < \widetilde{f}$ then
11.               $\widetilde{f} := \overline{F}(m(U^i))$
12.               $L_{work} :=$ CutOffTest($L_{work}, \widetilde{f}$)
13.           end
14.           if $w(f_U) < \epsilon$ then
15.               $L_{res} := L_{res} \cup (U, \underline{f_U})$
16.           else
17.               $L_{work} := L_{work} \cup (U, \underline{f_U})$
18.           end
19.       end
20.       if $L_{work} \neq \emptyset$ then $Y :=$ Head($L_{work}$)
21.   until $L = \emptyset$
22.   $Y :=$ Head($L_{res}$); $f^* := [\underline{f_Y}, \widetilde{f}]$
23.   return $L_{res}, f^*$
```

restricted for cases when the INTLAB based algorithm was able to stop within 10 minutes. Otherwise the test problem set is the same as those in other extensive numerical studies, such as Csendes [28, 29].

According to the paper (Csendes and Pál [32]), the results are summarized in Tables 3.1 and 3.2. The problem names are abbreviated as usual, e.g. S5 stands for Shekel-5, Sch3.2 for Schwefel 3.2, and R4 for Ratz-4 (cf. Csendes [28]). The first two columns give the problem names and their dimension. The listed efficiency indicators are the number of iterations necessary (abbreviated as NIT), the number of objective function evaluations (NFE), the maximal length of the working list (MLL), and the required CPU time in seconds (CPU).

As compared to the numerical study published in Pál and Csendes [83] and presented later in Section 3.3.3, where the stopping criterion parameter ϵ was set to 10^{-8}, now we stopped the subdivision when the width of the inclusion function at the actual subinterval was less than 0.01. Our present results are then obviously much weaker than the earlier published ones, but it is no wonder regarding that in the other case first and second derivative information was utilized as well.

Most of the efficiency indicators have the same or very similar values for the two implementations. We discuss here just the larger and systematic differences. The most significant change is definitely in the CPU time needed: the INTLAB based implementa-

Table 3.1: The numerical comparison of the C-XSC and the INTLAB code. Dim stands for the dimension of the problem, NIT for the number of iterations, and NFE for the number of objective function evaluations.

		Old, C-XSC code		New, INTLAB code	
Problem	Dim	NIT	NFE	NIT	NFE
S5	4	84	307	83	305
S7	4	259	864	259	864
S10	4	310	1,016	313	1025
THCB	2	5,591	16,779	5,591	16,779
BR	2	149	480	149	480
RB2	2	75	250	74	247
RB5	5	2,339	7,063	2,339	7,063
L8	3	21	81	21	81
L9	4	28	109	28	109
L10	5	35	137	35	137
L11	8	141	477	141	477
L12	10	412	1,455	412	1,455
L13	2	22	81	22	81
L14	3	35	131	35	131
L15	4	52	194	52	194
L16	5	72	270	72	270
L18	7	614	2,100	614	2,100
Schw2.1	2	308	933	308	933
Schw3.1	3	32	119	31	117
Schw2.5	2	72	232	72	232
Schw2.14	4	924	3,011	924	3,011
Schw2.18	2	5,623	17,093	5,623	17,093
Schw3.2	3	110	355	106	342
Schw3.7_5	5	191	605	191	605
Griew7	7	216	729	216	729
R4	2	1,547	5,137	1,547	5,137
R5	3	355	1,235	355	1,235
R6	5	1,939	6,695	1,939	6,695

Table 3.2: The numerical comparison of the C-XSC and the INTLAB code. Dim stands for the dimension of the problem, MLL for the maximal list length required, and CPU for the CPU time needed in seconds.

		Old, C-XSC code		New, INTLAB code	
Problem	dim	MLL	CPU	MLL	CPU
S5	4	14	0.02	13	14.20
S7	4	43	0.06	43	55.78
S10	4	58	0.17	55	93.73
THCB	2	1,128	0.36	1,128	178.58
BR	2	18	0.00	18	6.64
RB2	2	10	0.00	10	1.72
RB5	5	56	0.33	56	167.84
L8	3	8	0.00	8	1.94
L9	4	11	0.00	11	3.41
L10	5	14	0.00	14	5.22
L11	8	23	0.10	23	28.23
L12	10	44	0.71	44	111.63
L13	2	7	0.00	7	1.45
L14	3	10	0.00	10	3.09
L15	4	13	0.00	13	5.69
L16	5	16	0.01	16	9.55
L18	7	67	0.35	67	98.92
Schw2.1	2	44	0.00	44	11.63
Schw3.1	3	7	0.00	6	1.89
Schw2.5	2	7	0.00	7	1.44
Schw2.14	4	82	0.06	82	32.19
Schw2.18	2	678	0.48	678	104.73
Schw3.2	3	13	0.00	12	3.80
Schw3.7_5	5	32	0.01	32	6.59
Griew7	7	43	0.07	43	27.44
R4	2	348	0.11	348	51.39
R5	3	70	0.03	70	29.44
R6	5	226	0.48	226	264.31

tion requires on the average ca. 543 times more time to reach basically the same result. The ratios differ from 157 to 981, and the median of them is 523. As a rule, these figures are somewhat smaller than those measured for a more sophisticated algorithm variant based on the inclusion functions of the gradient and the Hessian as well (see Pál and Csendes [83] and later in Section 3.3.3). The highest ratio values are related to cases when the CPU time for the C-XSC version were hardly measurably low. It is also worth mentioning that the lowest ratios belong to those test problems that required more computation. The reason for this drop in speed is that MATLAB works in interpreter mode, and thus it is no wonder that a machine code program produced by a compiler can reach better times. On the other hand we have to add that we had a well readable, but less optimized coding, and there remained much to improve exploiting the vectorization feature of MATLAB. The bottom line of this comparison is that although the easy use of MATLAB has its price in speed, still for practical problems the INTLAB based interval global optimization method can be a useful modeling tool for early phases of optimization projects.

Since the number of iterations, objective function evaluations, and maximal working list lengths are identical for the two algorithms for the majority of test problems, we can certainly conclude that the algorithms are equivalent, and there cannot be significant algorithmic differences. In the remaining cases the slightly changing indicators are caused by the different realizations of the rounding and other hardware depending statements and functions. A smaller part of the CPU time differences is also due to the quicker but less precise interval operations and functions provided by Profil/BIAS.

3.3.3 An advanced procedure

3.3.3.1 The algorithm and its implementation

We describe a new implementation of an interval optimization algorithm which uses more advanced techniques. The algorithm implemented in MATLAB that uses the INTLAB package supporting interval calculations and automatic differentiation solves the bound constrained global optimization problem. The method itself is a simplified version of those interval techniques much investigated in the past, which were first developed from the global optimization algorithm of the Numerical Toolbox for Verified Computing (see Hammer et al. [46]).

The algorithm applies the most common accelerating devices: the cutoff test, the concavity test, the monotonicity test, and the interval Newton step. Beyond natural interval extension (based on naive interval arithmetic), a simple centered form inclusion function is also applied. Once the inclusion of the gradient is available, the intersection of these inclusion functions proved to be a good quality estimation of the range of the objective function.

We use also multisection and advanced subdivision direction selection (see Kearfott [62]), albeit without those based on the pf^* heuristic algorithm parameter (see Csendes [28]). Multisection means this time that each interval will be subdivided into three subintervals according to the most promising two coordinate directions. The subdivision directions are determined according to the well-tested and effective C subdivision direction selection rule (also used in Csendes [28], and Kearfott [62]). The algorithm solves also one-dimensional problems. For the latter problems the described multisection technique is substituted by greedy bisection made again on the basis of the C rule.

During the implementation we have followed closely the C-XSC code which was developed for bound constrained global optimization by Mihály Csaba Markót based on the algorithm documented in Markót et al. [68]. The control structures of the two algorithms are identical, while the vectorial array statements of MATLAB were applied wherever possible. The MATLAB/INTLAB based interval global optimization algorithm is available as a part of the GLOBAL package and it can be downloaded from

`www.inf.u-szeged.hu/~csendes/reg/regform.php`.

The comprised package contains all necessary files, a suitable directory structure and also a testing environment.

The advanced interval branch and bound method steps are listed in Algorithm 11. The algorithm takes as input the objective function f, the initial box X and a threshold value ϵ for the stopping criteria.

In line 2, the result list (L_{res}), the working list (L_{work}) and the temporary list (L_{work}) are initialized, while in line 3 an upper bound of global minimum value $\widetilde{f}$ is computed. The main iteration starts (line 4) with an optimal direction selection procedure and with a trisection of Y, getting three new boxes. Between lines 7 and 17 we iterate over the multisected boxes $U^i, i = 1...3$ and apply a monotonicity test (line 8), a range check (line 9) and a cut-off test (line 14). If the actual box is not discarded, it is stored in the L_{temp} list (line 16).

If the temporary list contains just one box (line 18), then we apply a concavity test (line 20), and a Newton step (line 22) on it, otherwise we put all new boxes in the result list (line 43) or in the working list (line 45). Note that the boxes are stored in the lists sorted in nondecreasing order with respect to the lower bounds of the inclusion function and in decreasing order with respect to the ages of the boxes.

The interval Newton step results in p boxes, to which we apply a monotonicity test (line 24) and a cut-off test (line 28). If the inclusion width of the actual interval V is less than ϵ, then the box is stored in the result list (line 32), otherwise in the working list (line 34). The ϵ value used in our algorithm was 10^{-8}.

The steps described previously are repeated until the working list becomes empty (line 50). When the main iteration stops, we compute a final enclosure (line 51) for the global minimum value and return L_{res} and f^* (line 52).

We mention that the algorithm locates all global minimizer points.

3.3.3.2 Computational test and comparison

The numerical comparison aimed to clear whether the new implementation is capable to deliver similar quality results as the old one, and to measure the efficiency in terms of the usual indicators. Hence, we have completed a computational test, and compared the efficiency and the results of the INTLAB implementation to that of a C-XSC, BIAS, and Profil based procedure (see Markót et al. [68]).

For the test we used INTLAB version 5.4, MATLAB R2007a, and a PC with 1 Gbyte RAM and a 3 GHz Pentium-4 processor. The test problems included all the standard

Algorithm 11. The advanced bound constrained interval global optimization algorithm

```
function AdvancedGOP ($f$, $X$, $\epsilon$)
  $L_{res} := \emptyset$; $L_{work} := \emptyset; L_{temp} := \emptyset$
  $Y := X$; $\widetilde{f} := \overline{F}(m(X))$
  repeat
        OptimalComponents($Y, k_1, k_2$)
        Trisection($Y, k_1, k_2, U^1, U^2, U^3$)
        for $i := 1$ to 3 do
              if MonotonicityTest($\nabla F(U^i)$) then continue
              $f_U := F(U^i)$
              if $\widetilde{f} < \underline{f_U}$ then continue
              $f_U := f_U \cap$ CenteredForm($U^i, \nabla F(U^i)$)
              if $\overline{F}(m(U^i)) < \widetilde{f}$ then
                    $\widetilde{f} := \overline{F}(m(U^i))$
                    $L_{work} :=$ CutOffTest($L_{work}, \widetilde{f}$)
              end
              if $\widetilde{f} >= \underline{f_U}$ then $L_{temp} := L_{temp} \cup (U^i, \underline{f_U})$
        end
        if length($L_{temp}$) $= 1$ then
              $U :=$ Head($L_{temp}$)
              if  not ConcavityTest($\nabla^2 F(U)$) then
                    $U :=$ Head($L_{temp}$)
                    NewtonStep($f, U, \nabla^2 F(U), V, p$)
                    for $i := 1$ to $p$ do
                          if MonotonicityTest($\nabla F(U^i)$) then continue
                          $f_V := f(V^i) \cap$ CenteredForm($V^i, \nabla F(V^i)$)
                          if $\overline{F}(m(V^i)) < \widetilde{f}$ then
                                $\widetilde{f} := \overline{F}(m(V^i))$
                                $L_{work} :=$ CutOffTest($L_{work}, \widetilde{f}$)
                          end
                          if $\widetilde{f} >= \underline{f_V}$ then
                                if $w(f_V) < \epsilon$ then
                                      $L_{res} := L_{res} \cup (V^i, \underline{f_V})$
                                else
                                      $L_{work} := L_{work} \cup (V^i, \underline{f_V})$
                                end
                          end
                    end
              end
        else
              while $L_{temp} \neq \emptyset$ do
                    $U :=$ Head($L_{temp}$)
                    if $w(f_U) < \epsilon$ then
                          $L_{res} := L_{res} \cup (U, \underline{f_U})$
                    else
                          $L_{work} := L_{work} \cup (U, \underline{f_U})$
                    end
              end
        end
        if $L_{work} \neq \emptyset$ then $Y :=$ Head($L_{res}$)
  until $L_{work} = \emptyset$
  $Y :=$ Head($L_{res}$); $f^* := [\underline{f_Y}, \widetilde{f}]$;
  return $L_{res}, f^*$
```

Table 3.3: The numerical comparison of the C-XSC and the INTLAB code. Dim stands for the dimension of the problem, NIT for the number of iterations, NFE for the number of objective function evaluations, and NGE for the number of gradient evaluations.

		Old, C-XSC code			New, INTLAB code		
Problem	Dim	NIT	NFE	NGE	NIT	NFE	NGE
S5	4	16	126	86	16	126	86
S7	4	18	129	84	17	121	78
S10	4	18	126	81	17	123	78
H3	3	42	184	135	42	184	135
H6	6	217	1,014	735	220	1,038	756
GP	2	2,351	15,314	9,430	2,351	15,319	9,427
SHCB	2	130	694	455	130	694	455
THCB	2	56	327	229	56	327	229
BR	2	52	282	200	52	282	200
RB	2	43	263	172	43	263	172
RB5	5	612	4,907	3,685	607	4,878	3,664
L3	2	293	1,890	1,301	293	1,890	1,301
L5	2	88	578	397	88	578	397
L8	3	11	80	55	11	80	55
L9	4	16	112	74	16	115	77
L10	5	19	143	97	19	143	97
L11	8	29	225	152	29	225	152
L12	10	34	282	194	34	282	194
L13	2	12	72	46	12	72	46
L14	3	15	104	66	15	104	66
L15	4	20	135	86	20	135	86
L16	5	19	142	88	19	142	88
L18	7	27	206	130	27	206	130
Schw2.1	2	226	1,312	951	226	1,312	951
Schw3.1	3	14	91	61	14	91	61
Schw2.5	2	53	307	216	53	307	216
Schw2.14	4	369	2,600	1,820	408	2,780	1,913
Schw2.18	2	51	284	201	51	284	201
Schw3.2	3	33	201	135	25	164	110
Schw3.7_5	5	129	677	484	129	677	484
Schw3.7_10	10	7,566	35,385	25,771	7,566	35,385	25,771
Griew5	5	691	9,854	6,424	705	9,940	6,482
Griew7	7	40	272	163	40	272	163
R4	2	154	902	648	153	899	645
R5	3	173	1,555	1,174	173	1,555	1,174
R6	5	227	2,255	1,826	227	2,255	1,826
R7	7	380	4,437	3,711	380	4,437	3,711
R8	9	471	6,170	5,266	471	6,170	5,266
EX2	5	41,794	250,885	177,929	14,774	89,318	65,862

global optimization functions to be minimized, and basically all of those usually applied in comparing interval global optimization methods. The test function Schwefel 2.7 is missing from the study, and hence also from the tables. The reason for it is that this problem cannot be solved by the algorithms within reasonable time (less than 10 minutes). Otherwise the test problem set is the same as those in other extensive numerical studies, such as Csendes [28, 29].

According to the paper (Pál and Csendes [83]), the results are summarized in Tables 3.3 and 3.4. The problem names are abbreviated again as usual, e.g. S5 stands for Shekel-5, Sch3.2 for Schwefel 3.2, and R4 for Ratz-4 (cf. Csendes [28]). The first two columns give the problem names and their dimension. The listed efficiency indicators are the number of iterations necessary (abbreviated as NIT), the number of objective function evaluations (NFE), the number of gradient evaluations (NGE), the number of Hessian evaluations (NHE), the maximal length of the working list (MLL), and the required CPU time in seconds (CPU).

Most of the efficiency indicators have the same or very similar values for the two implementations. We discuss here again just the larger and systematic differences. The most significant change is definitely also this time in the CPU time needed: the INTLAB based implementation requires on the average ca. 700 times more time to reach basically the same result. The ratios differ from 165 to 2,106, and the median of them is 619. The highest ratio values are related to cases when the CPU time for the C-XSC version were hardly measurably low. It is also worth mentioning that the lowest ratios belong to those test problems that required more computation. The reason for this drop in speed is that MATLAB works in interpreter mode, and thus it is no wonder that a machine code program produced by a compiler can reach better times. On the other hand we have to add that we had a well readable, but less optimized coding, and there remained much to improve exploiting the vectorization feature of MATLAB.

Since the number of iterations, objective function evaluations, gradient calls, Hessian evaluations and maximal working list lengths are identical for the two algorithms for the majority of test problems, we can certainly conclude that the algorithms are equivalent, and there cannot be significant algorithmic differences. In the remaining cases the slightly changing indicators are caused by the different realizations of the rounding and other hardware depending statements and functions. This finding is also supported by the fact that the somewhat larger differences (ca. 24%, 18%, 18%, 8%, 22%, and ca. 65%, 64%, 63%, 64%, 18%, respectively for the first five indicators in the Tables 3.3 and 3.4) obtained for the test problems Schwefel-3.2 and EX2 can well be led back for the flatness of these functions. The better efficiency indicators obtained for the latter cases are in accordance with the fact that the outside rounding necessary for the verified reliable bounds on the range of the functions is more precise in the INTLAB implementation. A smaller part of the CPU time differences is also due to the quicker but less precise interval operations and functions provided by Profil/BIAS.

Summarizing our numerical results, we can state that the computational experiences confirm that the new implementation is in several indicators (e.g. number of function, gradient and Hessian evaluations, number of iterations, and memory complexity) in essence equivalent to that of the old one. The CPU time needed is as a rule by at least two order of magnitude higher for the INTLAB version – as it can be anticipated regarding the interpreter nature of MATLAB. However, further vectorization coding

Table 3.4: The numerical comparison of the C-XSC and the INTLAB code. Dim stands for the dimension of the problem, NHE for the number of Hessian evaluations, MLL for the maximal list length required, and CPU for the CPU time needed in seconds.

		Old, C-XSC code			New, INTLAB code		
Problem	dim	NHE	MLL	CPU	NHE	MLL	CPU
S5	4	7	10	0.01	7	10	10.14
S7	4	7	14	0.03	6	14	13.05
S10	4	6	16	0.03	6	17	18.56
H3	3	3	12	0.01	3	12	11.20
H6	6	25	69	0.33	27	69	113.47
GP	2	608	480	0.68	608	480	630.33
SHCB	2	25	51	0.01	25	51	21.06
THCB	2	22	19	0.00	22	19	8.11
BR	2	18	12	0.00	18	12	6.91
RB	2	15	11	0.00	15	11	3.17
RB5	5	411	77	0.45	410	73	220.30
L3	2	97	138	0.16	97	138	115.11
L5	2	28	31	0.04	28	31	41.06
L8	3	5	9	0.00	5	9	4.03
L9	4	6	14	0.01	6	14	7.52
L10	5	8	17	0.02	8	17	11.97
L11	8	10	30	0.10	10	30	29.13
L12	10	12	39	0.24	12	39	46.28
L13	2	3	9	0.00	3	9	2.39
L14	3	5	11	0.00	5	11	4.59
L15	4	6	17	0.01	6	17	7.44
L16	5	6	20	0.01	6	20	9.33
L18	7	8	26	0.05	8	26	18.22
Schw2.1	2	88	26	0.02	88	26	36.78
Schw3.1	3	5	6	0.00	5	6	2.67
Schw2.5	2	28	5	0.00	28	5	4.02
Schw2.14	4	179	78	0.09	190	65	76.53
Schw2.18	2	22	8	0.00	22	8	3.81
Schw3.2	3	12	9	0.00	11	7	3.34
Schw3.7_5	5	32	32	0.03	32	32	25.98
Schw3.7_10	10	1,024	1,024	11.35	1,024	1,024	2,585.11
Griew5	5	597	32	1.04	611	32	575.95
Griew7	7	7	52	0.05	7	52	20.63
R4	2	51	39	0.02	51	39	17.72
R5	3	109	43	0.10	109	43	75.22
R6	5	143	29	0.39	143	29	186.75
R7	7	257	47	1.59	257	47	526.50
R8	9	321	65	3.81	321	65	951.27
EX2	5	19,124	1,969	72.93	6,928	1,610	12,042.27

changes in the algorithm and in the objective functions may improve on that. In spite of the lower speed, the new interval global optimization methods can well be suggested as an early modeling and experimentation tool for the verified solution of bound constrained global optimization problems.

3.3.4 Improvements on the Newton step

In the solution of an unconstrained optimization problem the gradient of the objective function is zero. Therefore, we can apply an interval Newton method to solve the nonlinear system $\nabla f(x) = 0$ over a box X in which we seek a minimum. We do not want to spend too much effort to sharply bound a stationary point if it is not the global minimum. Therefore, we do not want to iterate the Newton method to convergence. Instead, we want to alternate a step of the method with other procedures that might prove that a given stationary point is not a global minimum.

Therefore, we apply one step of the extended interval Newton Gauss-Seidel method (see Alefeld and Herzberger [1]) to the gradient of the objective function over a box X. As a result, we get a reduced box or boxes that contains the minimum.

The application of the Newton step on each single interval would be costly likewise. Hence, it would be suitable to introduce a criterion in order to decide when to apply the Newton step. In our study Pál and Csendes [83], we used the criterion described in Markót et al. [68]. In this case we applied the Newton step if two of the trisected boxes were eliminated by other acceleration tools (see Algorithm 11, line 18).

Now we examine a new condition for applying the Newton step. In this case we apply an interval Newton step to each box which has a smaller width than a prescribed value. It is well known that the Newton step is more efficient when it is applied to a small box rather than to a large one. This is because the effect of dependence is less for small boxes. In the new condition we used 0.1 as the threshold value.

We have implemented an algorithm in MATLAB/INTLAB with the new condition in order to compare it with the old one. The algorithm is similar to Algorithm 11, presented in the previous section. The most important differences are the following: the old interval selection rule is changed to the new one: we select the subinterval which has the maximal pf^* parameter value and the algorithm stops at the first box which satisfies the stopping criterion. The latter approach is acceptable in many practical situations when it is not necessary to find all global minimizer points.

The algorithm with the new condition for the Newton step is given as Algorithm 12. The algorithm takes as input the objective function f, the initial box X and a threshold value ϵ for the stopping criteria.

In line 2, the working list (L_{work}) and the temporary list (L_{temp}) are initialized, while in line 3 an upper bound of global minimum value $\widetilde{f}$ is computed. The main iteration starts (line 4) with an optimal direction selection procedure and with a trisection of Y, getting three new boxes. Between lines 7 and 17 we iterate over the multisected boxes $U^i, i = 1, 2, 3$ and apply a monotonicity test (line 8), a range check (line 9) and a cut-off test (line 14). If the actual box is not discarded, it is stored in the L_{temp} list (line 16).

Between lines 18 and 36, we iterate over the boxes from the temporary list. If the current box width is less than 0.1 (line 20), we apply the concavity test and the Newton step on it, otherwise we store it in the working list.

The described steps are repeated until we find the first box with smaller width than ϵ (line 39). In this case a final enclosure (line 40) for the global minimum value is computed and we return Y and f^* (line 41).

We have completed a computational test in order to compare the new condition with the old one. For the test we used INTLAB version 5.5, MATLAB R2008a, and a PC with 2 Gbyte RAM and a 2 GHz Pentium-4, Core 2 Duo processor. The test problems include all functions tested with Algorithm 11, too. We tested the algorithm with the new condition for the Newton step and compared with the original one.

According to the paper (Pál and Csendes [84]), the results are summarized in Tables 3.7 and 3.8. The first two columns give the problem names and their dimension. The listed efficiency indicators are the number of iterations necessary (abbreviated as NIT), the number of objective function evaluations (NFE), the number of gradient evaluations (NGE), the number of Hessian evaluations (NHE), the maximal length of the working list (MLL), and the required CPU time in seconds (CPU). In the end of the tables the aggregated indicator values can be found and the average values, calculated on the full set of test functions. It is hard to compare the two conditions, based on the received indicator values, because they differ significantly one by one. Therefore, it is more practical to compare the sum and average values over the whole testbed corresponding to the two conditions.

Considering the sum of Hessian evaluations and the total running time, we can conclude that are less by 25 and 14 percent in the case of the new condition. In the case of the number of objective function and gradient evaluations we can observ a slight decrease (1% and 3%). If we take into consideration the fact that the EX2 and Schw3.7.10 functions are claimed together 80 percentages of the full runtime, it is worth to examine the results without these two functions. In this case we find that the new condition reduces the total running time by 6 minutes (i.e. by 30%) while the total number of Hessian evaluations can be reduced by 48%. The sum of the number of objective function and gradient evaluations are less by 6% and 14%, respectively (see also in Tables 3.5 and 3.6).

Table 3.5: The sum and average values without the EX2 and Schw3.7.10 functions of the original and the new condition for the Newton step. NIT stands for the number of iterations, NFE for the number of objective function evaluations, and NGE for the number of gradient evaluations.

	Original condition			New condition		
	NIT	NFE	NGE	NIT	NFE	NGE
Sum	2665	25,061	18,984	4,119 (154%)	23,609 (94%)	16,420 (86%)
Average	72	677	513	111 (154%)	638 (94%)	444 (86%)

Based on these results it can be stated that we managed to reduce the necessary computational time and the total number of Hessian evaluations significantly with the help of the new condition. This improvement cannot be observed on every single function, and in some cases (e.g. Schw3.7.10) we obtain higher values. This means that it is hard to find a threshold value to the new condition so that it yields improvement for every function. We will examine the adaptive setting of this value in the future.

Algorithm 12. The algorithm with the new condition for the Newton step

```
function AdvancedNewCondGOP ($f$, $X$, $\epsilon$)
  $L_{work} := \emptyset; L_{temp} := \emptyset$
  $Y := X;\ \widetilde{f} := \overline{F}(m(X))$
  repeat
        OptimalComponents($Y, k_1, k_2$)
        Trisection($Y, k_1, k_2, U^1, U^2, U^3$)
        for $i := 1$ to 3 do
              if MonotonicityTest($\nabla F(U^i)$) then continue
              $f_U := F(U^i)$
              if $\widetilde{f} < \underline{f_U}$ then continue
              $f_U := f_U \cap$ CenteredForm($U^i, \nabla F(U^i)$)
              if $\overline{F}(m(U^i)) < \widetilde{f}$ then
                    $\widetilde{f} := \overline{F}(m(U^i))$
                    $L_{work} :=$ CutOffTest($L_{work}, \widetilde{f}$)
              end
              if $\widetilde{f} >= \underline{f_U}$ then $L_{temp} := L_{temp} \cup (U^i, \underline{f_U})$
        end
        for $i := 1$ to length($L_{temp}$) do
              $U :=$ Head($L_{temp}$)
              if $w(U) < 0.1$ then
                    if not ConcavityTest($\nabla^2 F(U)$) then
                          NewtonStep($f, U, \nabla^2 F(U), V, p$)
                          for $i := 1$ to $p$ do
                                if MonotonicityTest($\nabla F(V^i)$) then continue
                                $f_V := F(V^i) \cap$ CenteredForm($V^i, \nabla F(V^i)$)
                                if $\overline{F}(m(V^i)) < \widetilde{f}$ then
                                      $\widetilde{f} := \overline{F}(m(V^i))$
                                      $L_{work} :=$ CutOffTest($L_{work}, \widetilde{f}$)
                                end
                                if $\widetilde{f} >= \underline{f_V}$ then $L_{work} := L_{work} \cup (V^i, \underline{f_V})$
                          end
                    end
              else
                    $L_{work} := L_{work} \cup (U, \underline{f_U})$
              end
        end
        if $L_{work} \neq \emptyset$ then
              $Y :=$ Head($L_{work}$)
              if $w(f_Y) < \epsilon$ then
                    $f^* := [\underline{f_Y}, \widetilde{f}]$
                    return $Y, f^*$
              end
        end
  until $L_{work} = \emptyset$
```

Table 3.6: The sum and average values without the EX2 and Schw3.7.10 functions of the original and the new condition for the Newton step. NHE stands for the number of Hessian evaluations, MLL for the maximal list length required, and CPU for the CPU time needed in seconds.

	Original condition			New condition		
	NHE	MLL	CPU	NHE	MLL	CPU
Sum	1,565	1,206	1,158	832 (53%)	1,508 (125%)	811 (70%)
Average	42	33	31	22 (52%)	41 (124%)	22 (71%)

3.3.5 Theoretical examination of the Newton step

As we have seen in the previous section, the interval based global optimization algorithm can be improved by using a well chosen threshold value in the new condition for applying the Newton step. However, there are cases when the Newton step with the new condition is not successful in the sense that the width of the interval does not decrease or the current interval will be divided into many subintervals. The latter case appears in the practice mostly and is not beneficial because we may achieve a similar result by using some subdivision, but with a much smaller expense. The efficiency of the Newton step may further decrease in higher dimensions, because in this case the number of new subintervals are growing exponentially.

We suppose that the inclusion function of the second derivative has the following form where the Newton step is not successful:

$$F''_{new}(X) = \left[\underline{F}''(X) - D' \cdot w(F''(X)), \overline{F}''(X) + D'' \cdot w(F''(X))\right], \tag{3.12}$$

where $F''(X)$ is an inclusion of the second derivative over X, and D', D'' are positive real numbers. Throughout this investigation we also assume that the interval X contains a local minimizer point, since the Newton step role is to reduce the box which contains a such a point. $F''(X)$ may be the range of the second derivative over X. Similar inclusion functions have been considered in Casado et al. [24, 25].

Depending on the D' and D'' values we can distinguish between symmetric ($D' = D''$) and non-symmetric ($D' \neq D''$) overestimation. In the following section, we investigate separately the two types of overestimation.

3.3.5.1 The symmetric case

In the present subsection we suppose that the inclusion function defined by (3.12) is symmetric, which means that $D' = D'' = D$ (see also in Pál and Csendes [84]). The following two theorems want to characterize the cases when the Newton step is not efficient.

Theorem 3.8. *For a given single-variable function f and interval X, with $w(X) < \epsilon$, there exists an inclusion function of the second derivative with symmetric overestimation for which the Newton step is not successful in the sense that the width of the interval X does not decrease, or it divides the interval X into many subintervals.*

Table 3.7: The numerical comparison of the original and the new condition for the Newton step. Dim stands for the dimension of the problem, NIT for the number of iterations, NFE for the number of objective function evaluations, and NGE for the number of gradient evaluations.

		Original condition			New condition		
Problem	Dim	NIT	NFE	NGE	NIT	NGE	NGE
S5	4	16	126	86	22	117	76
S7	4	17	121	78	22	120	76
S10	4	17	123	78	22	122	76
H3	3	23	147	99	14	82	51
H6	6	191	1,505	1,167	112	560	363
GP	2	76	458	229	53	717	415
SHCB	2	17	103	60	16	105	63
THCB	2	44	274	189	59	284	187
BR	2	44	250	177	71	360	256
RB	2	38	238	151	17	174	117
RB5	5	396	3,660	2,758	608	3,511	2,568
L3	2	47	293	170	32	191	103
L5	2	86	593	406	22	131	73
L8	3	11	80	55	20	98	67
L9	4	13	107	73	26	129	85
L10	5	15	125	86	33	161	106
L11	8	23	189	128	52	253	163
L12	10	30	254	175	65	315	202
L13	2	10	74	47	13	76	49
L14	3	15	120	77	22	121	76
L15	4	18	146	94	28	150	94
L16	5	19	142	88	29	162	97
L18	7	27	206	130	41	226	136
Schw2.1	2	113	804	580	168	758	557
Schw3.1	3	14	96	64	21	122	81
Schw2.5	2	50	293	205	34	161	114
Schw2.14	4	356	3,242	2,337	527	5,914	4,160
Schw2.18	2	3	21	13	19	95	63
Schw3.2	3	20	144	98	25	149	99
Schw3.7_5	5	45	309	208	108	517	364
Schw3.7_10	10	696	4,371	2,665	5,232	22,065	15,781
Griew5	5	25	190	117	53	263	163
Griew7	7	40	297	173	73	363	223
R4	2	35	210	125	21	134	80
R5	3	107	996	748	181	760	544
R6	5	140	1,516	1,221	339	1,409	1,018
R7	7	204	2,728	2,293	500	2,086	1,501
R8	9	320	4,881	4,201	651	2,713	1,954
EX2	5	9,279	59,605	44,126	5,774	42,338	32,161
Sum		12,640	89,037	65,775	15,125	88,012	64,362
Average		324	2,283	1,687	388	2,257	1,650

Table 3.8: The numerical comparison of the original and the new condition for the Newton step. Dim stands for the dimension of the problem, NHE for the number of Hessian evaluations, MLL for the maximal list length required, and CPU for the CPU time needed in seconds.

		Original condition			New condition		
Problem	Dim	NHE	MLL	CPU	NHE	MLL	CPU
S5	4	7	10	5.66	3	10	5.02
S7	4	6	14	7.27	3	14	7.00
S10	4	6	17	10.36	3	17	9.95
H3	3	11	16	5.09	3	13	2.69
H6	6	86	64	97.45	10	55	32.77
GP	2	0	153	9.25	56	175	16.63
SHCB	2	3	22	1.70	6	19	1.80
THCB	2	21	24	3.75	3	26	3.59
BR	2	18	10	3.44	18	12	4.91
RB	2	11	11	1.59	19	9	1.25
RB5	5	317	79	93.13	162	79	83.20
L3	2	8	57	10.06	2	53	6.22
L5	2	31	32	26.25	2	32	5.25
L8	3	5	9	2.34	2	9	2.72
L9	4	7	13	4.08	2	14	4.67
L10	5	8	15	5.95	2	17	7.13
L11	8	9	28	14.11	2	30	17.42
L12	10	11	36	23.89	2	39	27.03
L13	2	4	9	1.45	3	9	1.47
L14	3	7	12	3.16	3	12	3.08
L15	4	8	19	4.84	3	18	4.73
L16	5	6	20	5.56	3	20	6.03
L18	7	8	26	10.80	4	26	11.31
Schw2.1	2	53	25	12.50	17	31	11.36
Schw3.1	3	5	6	1.58	7	6	1.95
Schw2.5	2	27	4	2.13	5	7	1.13
Schw2.14	4	216	123	47.98	455	442	88.22
Schw2.18	2	1	4	0.16	2	11	0.64
Schw3.2	3	11	7	1.66	9	8	1.66
Schw3.7_5	5	24	32	7.13	13	32	10.52
Schw3.7_10	10	192	818	183.59	28	1,024	873.70
Griew5	5	7	28	5.94	1	28	7.95
Griew7	7	8	58	12.48	1	51	15.02
R4	2	6	36	2.23	6	24	1.44
R5	3	71	57	27.20	0	23	18.97
R6	5	100	30	71.72	0	25	59.06
R7	7	168	41	184.78	0	47	122.38
R8	9	270	59	429.48	0	65	204.88
EX2	5	3,802	388	4,390.09	4,284	145	3,182.41
Sum		6,777	2,600	5,732	5,111	2,677	4,867
Average		174	67	147	132	69	125

Proof. In the case of a single-variable function $f(x)$, the Newton iteration can be described by the equations

$$N(X^{(k)}, \widetilde{x}^{(k)}) = \widetilde{x}^{(k)} - \frac{f'(\widetilde{x}^{(k)})}{F''(X^{(k)})}, \tag{3.13}$$

$$X^{(k+1)} = X^{(k)} \cap N(X^{(k)}, \widetilde{x}^{(k)}),\ k = 1, 2, \ldots, \tag{3.14}$$

where $\widetilde{x}^{(k)} = \text{mid}(X^{(k)})$, f' is the first derivative, and F'' is the inclusion of the second derivative.

Let $X = [a, b]$ be an interval such that $w(X) < \epsilon$ and let $F''(X) = [c, d]$. The Newton step is not successful if the inclusion of the second derivative (equation 3.13) contains the zero and after the intersection (equation 3.14) we obtain two new subintervals. This result is not useful for us because a similar result can also be archived by using a simple bisection, but with a much less expense. The question is, how much the overestimation should be in order to get two intervals after the intersection and the inclusion of the second derivative should contain the zero.

We further suppose that $0 \notin F''(X)$, otherwise it is obvious that the interval X will be subdivided. Using the notation $F''_{new}(X) = [c', d']$, the problem is now to find the values of c' and d' such that $0 \in F''_{new}(X)$ holds. Applying the previous notations, the Newton operator has the form

$$N(X, \widetilde{x}) = \widetilde{x} - \frac{f'(\widetilde{x})}{F''_{new}(X)} = \widetilde{x} - \frac{f'(\widetilde{x})}{[c', d']}. \tag{3.15}$$

Case 1: when $f'(\widetilde{x}) > 0$. Using the extended interval arithmetic we find that

$$N(X, \widetilde{x}) = \left(-\infty, \widetilde{x} - \frac{f'(\widetilde{x})}{d'}\right] \cup \left[\widetilde{x} - \frac{f'(\widetilde{x})}{c'}, \infty\right). \tag{3.16}$$

As a result of the $X \cap N(X, \widetilde{x})$ operation, we get two intervals if the following inequalities hold:

$$b > \widetilde{x} - \frac{f'(\widetilde{x})}{c'}, \tag{3.17}$$

$$a < \widetilde{x} - \frac{f'(\widetilde{x})}{d'}. \tag{3.18}$$

By (3.17) and (3.18) we obtain:

$$c' < -2 \cdot \frac{f'(\widetilde{x})}{w(X)}, \tag{3.19}$$

$$d' > 2 \cdot \frac{f'(\widetilde{x})}{w(X)}. \tag{3.20}$$

From (3.19), (3.20) and (3.12) it follows:

$$c - D \cdot w(F''(X)) < -2 \cdot \frac{f'(\widetilde{x})}{w(X)}, \tag{3.21}$$

$$d + D \cdot w(F''(X)) > 2 \cdot \frac{f'(\widetilde{x})}{w(X)}. \tag{3.22}$$

By (3.21) and (3.22) we obtain the following inequalities:

$$D > D_1 = \frac{2 \cdot f'(\widetilde{x}) + c \cdot w(X)}{w(X) \cdot w(F''(X))}, \tag{3.23}$$

$$D > D_2 = \frac{2 \cdot f'(\widetilde{x}) - d \cdot w(X)}{w(X) \cdot w(F''(X))}. \tag{3.24}$$

Consequently, if $D > \max(D_1, D_2)$ we get two new intervals as a result of the Newton step.

Case 2: when $f'(\widetilde{x}) < 0$. Using the extended interval arithmetic we obtain

$$N(X, \widetilde{x}) = \left(-\infty, \widetilde{x} - \frac{f'(\widetilde{x})}{c'}\right] \cup \left[\widetilde{x} - \frac{f'(\widetilde{x})}{d'}, \infty\right). \tag{3.25}$$

Proceeding with a similar argument to the first case, we have

$$D > D_1 = \frac{-2 \cdot f'(\widetilde{x}) + c \cdot w(X)}{w(X) \cdot w(F''(X))}, \tag{3.26}$$

$$D > D_2 = \frac{-2 \cdot f'(\widetilde{x}) - d \cdot w(X)}{w(X) \cdot w(F''(X))}. \tag{3.27}$$

As a result, if $D > \max(D_1, D_2)$, then the Newton step is not successful.

■

Proposition 3.9. *If $0 \notin F''(X)$ holds, then the constructed numbers D_1 and D_2 in the proof of the Theorem 3.8 have different sign.*

Proof. During the proof of the Theorem 3.8 we have constructed the D_1 and D_2 numbers so that the interval $N(X, \widetilde{x})$ was the smallest one which intersection with the interval X resulted two different intervals. As we supposed that $F''(X)$ doesn't contain the zero, it is obvious that D_1 or D_2 must be positive in order to obtain an overestimation of F'' which contains the zero. On the other hand, the result of the Newton step is always a non empty interval because a local minimizer point is available in X.

In this proof we also distinguish between the cases when the gradient is positive or negative. At first, we suppose that $f'(\widetilde{x}) > 0$ and $X = [a, b]$. It is also true that $c > 0$ in $F''(X)$, because we considered a local minimizer point in X. The Newton operator using F'' in (3.15) is

$$N(X,\widetilde{x}) = \widetilde{x} - \frac{f'(\widetilde{x})}{F''(X)} = \left[\widetilde{x} - \frac{f'(\widetilde{x})}{c}, \widetilde{x} - \frac{f'(\widetilde{x})}{d}\right]. \tag{3.28}$$

Since $X \cap N(X,\widetilde{x}) \neq \emptyset$, then it follows that $\widetilde{x} - \frac{f'(\widetilde{x})}{d} > a$ must holds (see also in Figure 3.4). Now if we increase the value of d ($D_2 > 0$), then the value of $\widetilde{x} - \frac{f'(\widetilde{x})}{d}$ is also increasing and the left hand side interval in (3.16) will be also larger. If the value of d decreases ($D_2 < 0$), then the value of $\widetilde{x} - \frac{f'(\widetilde{x})}{d}$ also decreases. As we computed the smallest interval in the proof of the Theorem 3.8, thus D_2 must be negative.

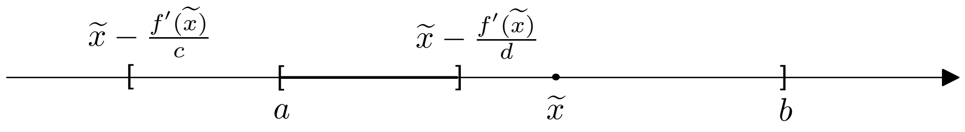

$\widetilde{x} - \frac{f'(\widetilde{x})}{c}$
$\widetilde{x} - \frac{f'(\widetilde{x})}{d}$
a
$\widetilde{x}$
b

Figure 3.4: The result of the Newton step if $f'(x) > 0$.

In the second case, if $f'(\widetilde{x}) < 0$, then the Newton operator has the following form

$$N(X,\widetilde{x}) = \widetilde{x} - \frac{f'(\widetilde{x})}{F''(X)} = \left[\widetilde{x} - \frac{f'(\widetilde{x})}{d}, \widetilde{x} - \frac{f'(\widetilde{x})}{c}\right]. \tag{3.29}$$

Since $X \cap N(X,\widetilde{x}) \neq \emptyset$, then it follows that $\widetilde{x} - \frac{f'(\widetilde{x})}{d} < b$ must holds (see also in Figure 3.5). Now if we increase the value of d ($D_2 > 0$), then the value of $\widetilde{x} - \frac{f'(\widetilde{x})}{d}$ decreases and the right hand side interval in (3.25) will be larger. If the value of d decreases ($D_2 < 0$), then the value of $\widetilde{x} - \frac{f'(\widetilde{x})}{d}$ increases. As we computed the smallest interval in the proof of the Theorem 3.8, thus D_2 must be negative.

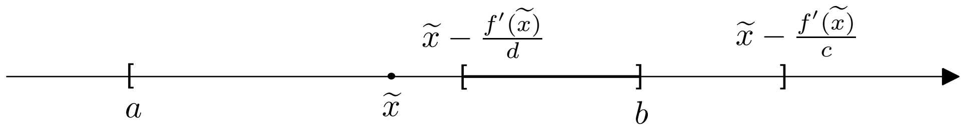

$\widetilde{x} - \frac{f'(\widetilde{x})}{d}$
$\widetilde{x} - \frac{f'(\widetilde{x})}{c}$
a
$\widetilde{x}$
b

Figure 3.5: The result of the Newton step if $f'(x) < 0$.

■

Remark 3.10. *There is no sense to have $D_2 < 0$ from two reasons. Firstly, the $F''_{new}(X)$ cannot be an overestimation of the $F''(X)$ using D_2, which is inacceptable in our case (where $F''(X)$ is the range of the second derivative). Secondly, the resulted interval may not contain the local minimizer point. The constructed value of D in the proof of the Theorem 3.8 guarantees that the resulted interval contains the local minimizer point. We supposed that $0 \notin F''(X)$ holds, otherwise we may have $D_1 < 0$ and $D_2 < 0$ if $F''(X)$ is too large.*

The presented remarks are demonstrated by the following example:

Example 3.11. *In the case of the function $f(x) = \frac{1}{20}(x+4)(x+2)(x+1)(x-1)(x-3)+2$, over the interval $X = [-1.7, -1.4]$ we find that $D > D_1 = 0.63$ and $D > D_2 = -1.336$. Now if we use $D' = 0.64$ and $D'' = -1.33$ in (3.12), we obtain (according to*

(3.14)) $[-1.70, -1.69] \cup [-1.40, -1.39]$ *which doesn't contain the local minimizer point -1.52. Now if we use* $D' = D'' = D = 0.64$*, then the union of the resulted intervals* $[-1.70, -1.69] \cup [-1.53, -1.39]$ *contains the local minimizer point.*

Now we examine the case when the Newton step is successful in the sense that we obtain a narrower interval than the input one. According to the first case from Theorem 3.8, this may happen if one of the two inequalities (3.23) and (3.24) or both of them doesn't hold. At first, we assume that $D \le D_1$ and $D > D_2$ hold, where $D_1 > 0$ and $D_2 < 0$. Because D must be positive then we have $D \in (0, D_1]$ (the second inequality can be omitted). Expanding the inequality $D \le D_1$ we obtain

$$D \le D_1 = \frac{2 \cdot f'(\widetilde{x}) + c \cdot w(X)}{w(X) \cdot w(F''(X))}. \tag{3.30}$$

Now if we assume that the overestimation parameter D is known in (3.30), we can specify a threshold value ϵ which guarantees the success of the Newton step in the sense that we receive a narrower interval than the input one. From (3.30) it follows that the requested condition is

$$w(X) \le \frac{2 \cdot f'(\widetilde{x})}{D \cdot w(F''(X)) - c}. \tag{3.31}$$

Thus, if

$$\epsilon = \frac{2 \cdot f'(\widetilde{x})}{D \cdot w(F''(X)) - c},$$

holds, the Newton step returns a narrower interval than the input one. In the second case we have $D > D_1$ and $D \le D_2$, which doesn't provide useful information due to the latter inequality. The third case ($D \le D_1$ and $D \le D_2$) is similar to the first one. We obtain a similar statement if $f'(\widetilde{x}) < 0$ (Case 2 in the proof of Theorem 3.8) is supposed.

Example 3.12. *The Newton step is not successful for the function* $f(x) = \frac{1}{20}(x+4)(x+2)(x+1)(x-1)(x-3)+2$ *over the interval* $X = [-1.6, -1.4]$ *if* $D > 1.2994$*. Now if we apply the condition (3.31) for* $D = 1.2994$*, we find that if* $w(X) < 0.2$*, then the Newton step is successful. This result means that in case of intervals which contains the local minimizer point, has a smaller width than 0.2 and use the fixed D value in (3.12), the Newton step returns a narrower interval than the input one.*

An inclusion function $F(X)$ is isotone if $X \subseteq Y$ implies $F(X) \subseteq F(Y)$. Almost every inclusion function used in numerical procedures possess this property. The inclusion function F''_{new} defined in (3.12) is naturally isotone for each fixed value of D. However, the next statement says that if the value of D have to be defined dynamically so that the Newton step shouldn't be successful, then the resulting inclusion function will not be isotone.

Proposition 3.13. *The inclusion function* F''_{new} *with the smallest value of D, constructed in the proof of the Theorem 3.8, is not isotone.*

Proof. In order to prove this statement it is enough to find a counterexample. For example, consider the function $f(x) = \frac{1}{20}(x+4)(x+2)(x+1)(x-1)(x-3)+2$. The inclusion function of f defined by (3.12) and constructed according to the Theorem 3.8, is not isotone over the following intervals.

If $X_1 = [-1.60, -1.5]$, then $D > 3.7569$. A suitable value of D is 3.76 and $F''_{new}(X_1) = [-1.41, 7.46]$.

If $X_2 = [-1.65, -1.45]$, then $D > 1.3002$. A suitable value of D is 1.31 and $F''_{new}(X_2) = [-0.70, 6.81]$.

If $X_3 = [-1.7, -1.4]$, then $D > 0.6353$. A suitable value of D is 0.64 and $F''_{new}(X_3) = [-0.46, 6.67]$.

Consequently, $X_1 \subset X_2 \subset X_3$ and $F''_{new}(X_3) \subset F''_{new}(X_2) \subset F''_{new}(X_1)$, thus F''_{new} is not isotone.

■

The essential meaning of the Proposition 3.13 is that small argument intervals require large excess width to make the Newton step ineffective. In other words, by choosing a rather small value of the decision parameter ϵ, the success of the Newton step may be increased. The above result may appear somewhat surprising since usually the isotonicity is fulfilled by the inclusion functions. Nevertheless, the following statement says that we always can choose values for D such that the resulted inclusion function will be isotone.

Proposition 3.14. *The values of D can always be chosen so that the inclusion function F''_{new} constructed in the Theorem 3.8 will be isotone.*

Proof. From the proof of the Theorem 3.8 it follows that the constructed D value can be chosen arbitrary large. Therefore, for every intervals X_1 and X_2 where $X_1 \subseteq X_2$ we can choose the values of D such that $F''_{new}(X_1) \subseteq F''_{new}(X_2)$ holds.

■

The above proposition is demonstrated by the following example:

Example 3.15. *Consider again the function $f(x) = \frac{1}{20}(x+4)(x+2)(x+1)(x-1)(x-3)+2$ and intervals X_1, X_2 and X_3.*

If $X_1 = [-1.60, -1.5]$, then $D > 3.7569$. A suitable value of D is 3.76 and $F''_{new}(X_1) = [-1.41, 7.46]$.

If $X_2 = [-1.65, -1.45]$, then $D > 1.3002$. A suitable value of D is 2 and $F''_{new}(X_2) = [-2.16, 8.20]$.

If $X_3 = [-1.7, -1.4]$, then $D > 0.6353$. A suitable value of D is 1.5 and $F''_{new}(X_3) = [-3.18, 9.38]$.

Consequently, $X_1 \subset X_2 \subset X_3$ and $F''_{new}(X_1) \subset F''_{new}(X_2) \subset F''_{new}(X_3)$. Obviously the above steps can be repeated for any two intervals X_1 and X_2 ($X_1 \subset X_2$).

Theorem 3.16. *For a given multivariate function f and X interval, with $w(X) < \epsilon$, there exists an inclusion function of the Hessian for which the Newton step is not successful in the sense that the length of the interval X does not decrease or it divides the interval X into many subintervals.*

Proof. The multivariate Newton method can be described by the equations:

$$G(\widetilde{x}^{(k)}) + H(X^{(k)})(N(X^{(k)}, \widetilde{x}^{(k)}) - \widetilde{x}^{(k)}) = 0, \tag{3.32}$$

$$X^{(k+1)} = X^{(k)} \cap N(X^{(k)}, \widetilde{x}^{(k)}),\ k = 1, 2, \ldots, \tag{3.33}$$

where $\widetilde{x}^{(k)} = \text{mid}(X^{(k)})$, G is the inclusion function of the gradient, and H is the inclusion function of the Hessian matrix.

The operator N is that developed by Hansen and Sengupta (see Hansen and Sengupta [54]), and it uses a Gauss-Seidel procedure. Preconditioning of the equation (3.32) with C the midpoint inverse of $H(X)$ gives

$$C \cdot H(X)(N(X, \widetilde{x}) - \widetilde{x}) = -C \cdot G(\widetilde{x}).$$

By defining

$$M = C \cdot H(X), \quad b = C \cdot G(\widetilde{x}),$$

the interval Gauss-Seidel procedure proceeds component by component to give the iteration

$$N(X^{(k)}, \widetilde{x}^{(k)}) = \widetilde{x}_i^{(k)} - \frac{b_i + \sum_{j=1}^{i-1} M_{ij}(X_j^{(k+1)} - \widetilde{x}_j^{(k+1)}) + \sum_{j=i+1}^{n} M_{ij}(X_j^{(k)} - \widetilde{x}_j^{(k)})}{M_{ii}}, \tag{3.34}$$

$$X_i^{(k+1)} = X_i^{(k)} \cap N(X^{(k)}, \widetilde{x}^{(k)}), \tag{3.35}$$

where $\widetilde{x}^{(k)} = \text{mid}(X^{(k)})$ and $k = 1, 2, \ldots$.

In this iteration, after the i-th component of $N(X^{(k)}, \widetilde{x}^{(k)})$ is computed using (3.34), the intersection (3.35) is performed. The result is then used to calculate subsequent components of $N(X^{(k)}, \widetilde{x}^{(k)})$.

At first, we consider the simple case when the inclusion of the Hessian matrix is used instead of the preconditioned version of it. Suppose that it has the following form:

$$H = \begin{pmatrix} H_{11} & H_{11} & \cdots & H_{1n} \\ H_{21} & H_{22} & \cdots & H_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ H_{n1} & H_{n2} & \cdots & H_{nn} \end{pmatrix},$$

where $H_{ij} : \mathbb{R}^n \to \mathbb{R}$, $i, j = 1, \ldots, n$ are also inclusion functions. We also suppose that $0 \notin H_{ii}$, $i = 1, \ldots, n$. Similarly to the equation (3.34), in the iteration we divide by a diagonal element of the H matrix. Hence, in this case we should determine the overestimation of the H_{ii}, $i = 1, \ldots, n$ inclusion functions so that at the end of the iteration should obtain more then one interval. On the other hand, the operator N in an iteration is similar to the equation (3.15). Although the inclusion function H_{ii} usually is not a single-variable function, still the value of D can be determined similarly to the Theorem 3.8. As a result of the described procedure for some i, the component

X_i consists of two intervals separated by an open set (gap). In this case the box X may be split normal to this coordinate direction. In order to avoid an exponential increase of the number of subboxes the further steps are applied to the hull of the subboxes in such cases. A splitting is therefore only done once during the iteration, vertical to the direction of the largest gap or two largest gaps at the end of the iteration.

Consequently, if for some i the component X_i is subdivided, then the box X will be splitted into at least two subboxes. Thus, the Newton step is not efficient in this case.

Next consider the case when the preconditioned Hessian matrix is used. The precondition aim to transform the original system to a system that is almost diagonally dominant. This kind of systems can be solved more efficiently with the Gauss-Seidel method. This case is very similar to the previous one. The main difference is that in an iteration we divide by M_{ii} instead of H_{ii}, where

$$M_{ii} = \sum_{j=1}^{n} c_{ij} H_{ji},\ i = 1, \dots, n,$$

and c_{ij}, $i, j = 1, \dots, n$ are the elements of the matrix C. Although M_{ii} is more complex than H_{ii} the construction is similar to the first case.

■

3.3.5.2 The general case

The whole construction from the symmetric case can be transformed for inclusion functions with non-symmetric overestimation. In this case we suppose that $D' \neq D''$ holds in (3.12).

Theorem 3.17. *For a given single-variable function f and interval X, with $w(X) < \epsilon$, there exists an inclusion function of the second derivative with non-symmetric overestimation for which the Newton step is not successful in the sense that the width of the interval X does not decrease, or it divides the interval X into many subintervals.*

Proof. The proof is similar to the Theorem 3.8. As a result, we find that $D' > D_1$ and $D'' > D_2$, where D_1 and D_2 are defined by (3.23), (3.24) (Case 1) and (3.26),(3.27) (Case 2), respectively.

As we have seen previously, D_1 or D_2 will be negative. In this case the $F''_{new}(X)$ cannot be an overestimation of the $F''(X)$, which is inacceptable in our case (where $F''(X)$ is the range of the second derivative). In order to avoid this, we modify the relative values D' and D'' of the overestimation in the following way:

$$D' > \max(0, D_1) \text{ and } D'' > \max(0, D_2).$$

Obviously, we may have $D' = 0$ or $D'' = 0$. In this case the lower bound or the upper bound of the $F''(X)$ will not be modified. The chosen values also guarantees that the obtained new inclusion function is an overestimation of the original one and the resulted interval after the Newton step contains the local minimizer point.

■

Proposition 3.18. *The inclusion function F''_{new} with the smallest values of D' and D'', constructed in the proof of the Theorem 3.17, is not isotone.*

Proof. The proof is similar to the Proposition 3.13. Consider the function $f(x) = x^3 - 2x$. The inclusion function of f defined by (3.12) and constructed according to the Theorem 3.17, is not isotone over the following intervals.

If $X_1 = [0.7, 0.9]$, then $D' > 4.1667$ and $D'' \geq 0$. A suitable value of D' and D'' is 4.2 and 0. $F''_{new}(X_1) = [-0.84, 5.4]$.

If $X_2 = [0.6, 1]$, then $D' > 1.6667$ and $D'' \geq 0$. A suitable value of D' and D'' is 1.67 and 0. $F''_{new}(X_2) = [-0.40, 6.0]$.

If $X_3 = [0.5, 1.1]$, then $D' > 0.9074$ and $D'' \geq 0$. A suitable value of D' and D'' is 0.91 and 0. $F''_{new}(X_3) = [-0.27, 6.6]$.

Consequently, $X_1 \subset X_2 \subset X_3$ and $F''_{new}(X_1) \subsetneq F''_{new}(X_2) \subsetneq F''_{new}(X_3)$, thus F''_{new} is not isotone. In this case the relation $F''_{new}(X_3) \subset F''_{new}(X_2) \subset F''_{new}(X_1)$ doesn't hold (see the example in the symmetric case) due to the isotonicity of the F'' (the upper bound of F''_{new} is equal with the upper bound of F''). ■

Proposition 3.19. *The values of D' and D'' can always be chosen so that the inclusion function F''_{new} constructed in the Theorem 3.17 will be isotone.*

Proof. The proof is similar to the Proposition 3.14. ■

The above proposition is demonstrated by the following example:

Example 3.20. *Consider again the function $f(x) = x^3 - 2x$ and intervals X_1, X_2 and X_3.*

If $X_1 = [0.7, 0.9]$, then $D' > 4.1667$ and $D'' \geq 0$. A suitable value of D' and D'' is 4.2 and 0. $F''_{new}(X_1) = [-0.84, 5.4]$.

If $X_2 = [0.6, 1]$, then $D' > 1.6667$ and $D'' \geq 0$. A suitable value of D' and D'' is 1.87 and 0. $F''_{new}(X_2) = [-0.88, 6.0]$.

If $X_3 = [0.5, 1.1]$, then $D' > 0.9074$ and $D'' \geq 0$. A suitable value of D' and D'' is 1.1 and 0. $F''_{new}(X_3) = [-0.96, 6.6]$.

Consequently, $X_1 \subset X_2 \subset X_3$ and $F''_{new}(X_1) \subset F''_{new}(X_2) \subset F''_{new}(X_3)$. Obviously the above steps can be repeated for any two intervals X_1 and X_2 ($X_1 \subset X_2$).

Theorem 3.21. *For a given multivariate function f and X interval, with $w(X) < \epsilon$, there exists an inclusion function of the Hessian for which the Newton step is not successful in the sense that the length of the interval X does not decrease or it divides the interval X into many subintervals.*

Proof. The proof is analogous to the Theorem 3.16. The only difference is that the D' and D'' values should be determined according to the Theorem 3.17. ■

3.4 Application in sensor network localization

3.4.1 Introduction

Location awareness is important for wireless sensor networks since many applications such as environment monitoring (animal habitat monitoring, bush fire surveillance, water quality monitoring, etc.), vehicle tracking and mapping, military applications (battlefield surveillance) depend on knowing the locations of sensor nodes.

Sensor network localization refers to the process of estimating the locations of sensors using measurements between neighboring sensors. In sensor network localization, it is assumed that a small fraction of sensors, called *anchors*, have a priori information about their positions. The coordinates of the anchor nodes may be obtained by using a global positioning system (GPS) or manual deployment at a point with known position. Most sensors do not have a priori information about their locations because it is often too expensive to include a GPS adapter in each sensor node or due to constraints on cost and complexity. These sensor nodes without a priori position information are referred to as the *non-anchor nodes* and their coordinates are to be estimated by sensor network localization algorithms. These algorithms differ in their assumptions about network configuration and mobility, the distribution of the calculation process and hardware capabilities. The location calculation can be done using centralized algorithms or distributed ones. In a centralized algorithm the sensor nodes transmit the data to a central computer, where calculation is performed to determine estimated location of each node. In distributed algorithms, there is no central computer. Each non-anchor node estimates its location using measurements between neighbouring nodes and the location estimates of its neighbours.

Considering hardware capabilities, two categories of localization methods can be distinguished: distance-based and connectivity based. Distance-based methods (also called range-based) use inter-sensor distance or angle measurements in location calculation. There are many approaches for the implementation of the centralized distance-based algorithms. The most important are the following: multidimensional scaling (see Shang et al. [106]), semidefinite programming (see Biswas and Ye [13]) and simulated annealing (see Kannan et al. [61], Niewiadomska-Szynkiewicz and Marks [79]). Connectivity-based (also called range-free) algorithms do not rely on measurement techniques. They use the connectivity information to derive the locations of the non-anchor nodes. The most important connectivity-based algorithms are hop-counting techniques. Examples of connectivity-based methods are presented in Niculescu and Nath [78], and Shang et al. [106].

In this thesis, we focus on localization methods based on distance measurements.

3.4.2 Mathematical formulation of the localization problem

The mathematical model of the distance-based general localization problem considered on the plan can be described as follows. Suppose we have a set of m anchor nodes with known positions $a_k \in \mathbb{R}^2$, $k = 1, \ldots, m$, and n non-anchor nodes $x_j \in \mathbb{R}^2$, $j = 1, \ldots, n$ with unknown locations. For each pair of two nodes in the sensor network, we define the Euclidean distance $d_{kj} = ||a_k - x_j||$ between anchors and non-anchors, and

$d_{ij} = ||x_i - x_j||$ between two non-anchors, $j = 1, \dots, n$ and $i \neq j$.

Based on the emitted signal, a sensor have the capability to estimate the distances from the neighboring nodes. The sets of neighboring nodes can defined as collections of nodes located within transmission ranges (radio ranges) of given nodes. Hence, for all anchors and non-anchors we define the following sets:

$$N_k = \{(k,j) : d_{kj} \leq r_k\}, \; j = 1, \dots, n,$$
$$N_i = \{(i,j) : d_{ij} \leq r_i\}, \; j = 1, \dots, n,$$

where r_k and r_i are radiuses (maximal transmission ranges) of the k-th anchor node and the i-th non-anchor node, respectively.

The measured values $\widetilde{d}_{kj}$ and $\widetilde{d}_{ij}$ of true distances d_{kj} and d_{ij} are produced by measurement methods described in Mao et al. [66]. These methods involve measurement errors, hence each distance value $\widetilde{d}_{kj}$ and $\widetilde{d}_{kj}$ represents the true distance corrupted with noise describing the uncertainty of the distance measurement,

$$\widetilde{d}_{kj} = d_{kj} + \xi_{kj}, \quad \widetilde{d}_{ij} = d_{ij} + \xi_{ij},$$

where ξ_{kj} and ξ_{ij} denote measurement errors.

Based on the above definitions, the general localization problem can be formulated in the following way: estimate the locations of non-anchor nodes $x_j \in \mathbb{R}^2$, $j = 1, \dots, n$ with unknown positions, if positions of anchor nodes $a_k \in \mathbb{R}^2$ and noisy distance measurements $\widetilde{d}_{kj}$, $\widetilde{d}_{ij}$ are known. The presented model can be formulated as an optimization problem with the aim to minimize the sum of errors in sensor positions for fitting the distance measurements.

3.4.3 Solution approaches

The previously described distance-based localization problem can be formulated as a quadratic optimization problem, and then transformed to a standard semidefinite programming problem. Such formulation is presented in Biswas and Ye [13]. This kind of optimization problems can be solved by interior point methods based on SDP solvers.

Another approach is to formulate the optimization problem with the following nonlinear performance function and apply global optimization methods to solve it:

$$\min_{\widehat{x}} \left\{ \sum_{k=1}^{m} \sum_{j \in N_k} \left(||a_k - \widehat{x}_j|| - \widetilde{d}_{kj} \right)^2 + \sum_{i=1}^{n} \sum_{j \in N_i} \left(||\widehat{x}_i - \widehat{x}_j|| - \widetilde{d}_{ij} \right)^2 \right\}, \tag{3.36}$$

where $\widehat{x}_i$ and $\widehat{x}_j$ denote, respectively, estimated positions of nodes i and j, $\widetilde{d}_{kj}$ and $\widetilde{d}_{ij}$ are measured distances between the pairs of nodes (k,j) and (i,j), and N_i, N_k are sets of neighboring nodes.

The most frequently used global optimization methods are the simulated annealing (see Kannan et al. [61], Mao et al. [66]) and genetic algorithms (see Zhang et al. [118]). There are hybrid localization methods (see Niewiadomska-Szynkiewicz and Marks [79]) which combine trilateration techniques and stochastic methods. These methods in the first phase provide an initial solution. The solution of the first phase is modified by applying stochastic optimization methods.

In our approach, we tried to find the global optimum of the objective function (3.36) using the INTLAB based global optimization tool. One reason for using it was to represent the uncertainty in the distance measurements with interval variables, but the obtained result doesn't provide useful information in the sense that the final interval was too wide. Therefore, we tried to solve the problem without incorporating the uncertainty into the intervals.

Finding the global optimum of the (3.36) objective function with the INTLAB based global optimizer is a time consuming task especially using hundreds of sensors. Hence, we tried to find an approximated solution using the trilateration technique. Trilateration is an approach to determine the position of an unknown node using three references located at known positions. If we have more than three nodes, it is called multilateration. As the inter-node distances are corrupted with noise, the trilateration has no unique solution. Hence, we are looking for that point, which minimizes the sum of distances from the nodes with know positions. There are many solution approaches to the trilateration like the quadratic equation method, Cayley-Menger determinant method, nonlinear least squares method. We used the last one combined with the INTLAB based global optimizer.

Trilateration can be applied in an iterative manner to determine the locations of all nodes in the following way: initially an unknown node, if possible, is located based on its neighbors by trilateration. After being aware of its location, it becomes a reference node to localize other unknown nodes in the subsequent localization process. This step continues iteratively, gradually turning the unknown nodes to known.

The noisy distance measurement degrades the quality of trilateration in the following aspects: uncertainty, non-consistency, ambiguity and error propagation has its effect. There are heuristic methods (see Savarese et al. [103], Yang and Liu [115]) which try to improve the quality of the trilateration. In our approach, we use four randomly selected reference nodes to determine the position of a node.

3.4.4 Numerical results

The solution of the localization problem largely depends on the number of anchor and non-anchor nodes and their positions, on the number of the neighboring nodes, on the error of the measured distances and on the value of the transmission range. In our numerical tests we used 200 nodes with randomly generated positions in a $[0, 1] \times [0, 1]$ square region. The proportion of anchor nodes was 10%. We assumed a fixed transmission range r for all nodes. The measured distance between nodes is blurred by introducing a Gaussian noise into the true distance in the following way:

$$\widetilde{d_{kj}} = d_{kj}(1 + randn() \cdot nf),$$

$$\widetilde{d_{ij}} = d_{ij}(1 + randn() \cdot nf),$$

where d_{kj} and d_{ij} are the true distances, $randn()$ is a Gaussian distributed random variable with 0 mean and variance 1 and nf is the noise factor with the value of 10%.

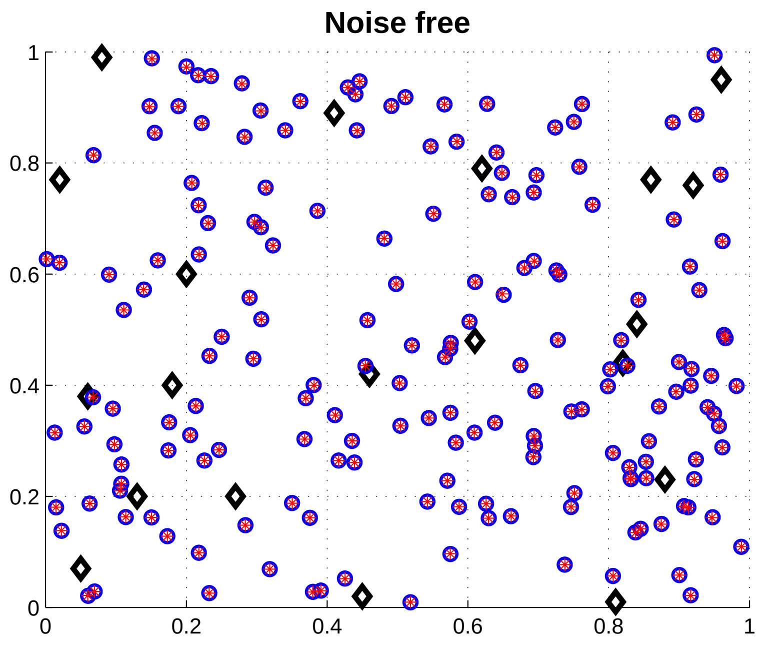


Figure 3.6: The iterative trilateration result with noise free distances.

For the performance evaluation we used the mean error between the estimated and the true position of non-anchor nodes, defined as

$$LE = \frac{1}{n} \sum_{i=1}^{n} \frac{(||\widehat{x}_i - x_i||)^2}{r^2} 100\%,$$

where LE is the localization error, x_i is the true position of sensor node i, $\widehat{x}_i$ is the estimated location of sensor node i and r is the transmission range.

In Figure 3.6, the true and estimated locations are shown in case when there is no error in the distance measurements. Figure 3.7 presents the cases when we use three (trilateration) and four neighboring nodes (multilateration) with noisy distances. The corresponding localization errors are 0.0005%, 66.7%, and 24.8%, respectively. The anchor nodes are marked with diamonds, the true positions of non-anchor nodes with circle while the estimated locations with stars. The localization error is denoted by lines connecting the real and estimated locations.

We evaluated the multilateration using some key metrics: the accuracy of the location estimates versus the anchor nodes number and radio range. The presented numerical results are the average of ten executions.

Figure 3.8(a) shows the impact of the number of anchor nodes and radio range on localization accuracy. As it can be observed, the localization error decreases as the number of anchor nodes increases. Using more anchor nodes it makes the localization easier but it also increases the deployment costs.

The transmission range determines the number of neighbors of a node (see Table 3.9) and may have a serious impact on localization accuracy. A relationship between transmission range vs. average connectivity (neighbor nodes number) is shown in Figure 3.8(b). In this case, as the radio range increases, the localization error decreases.

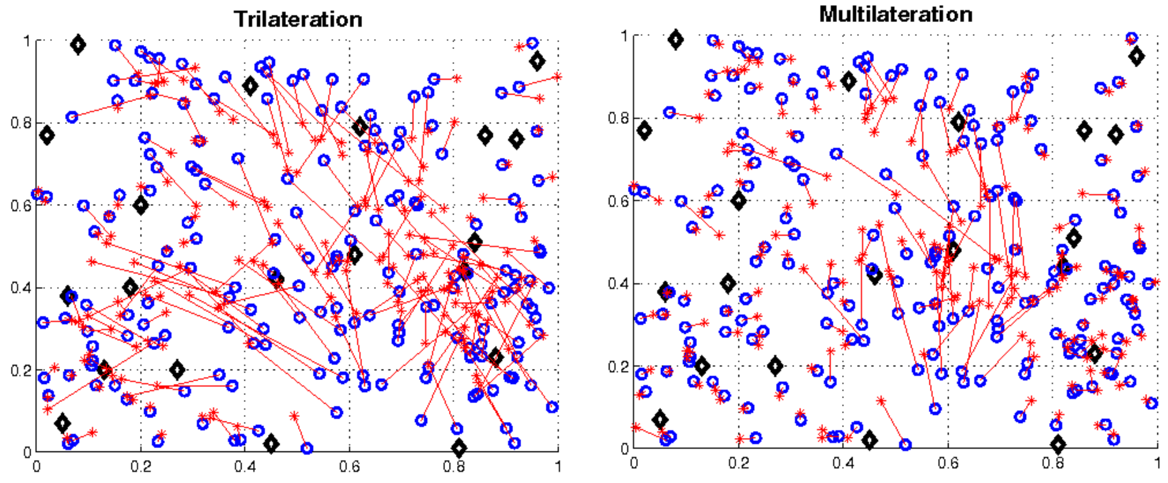
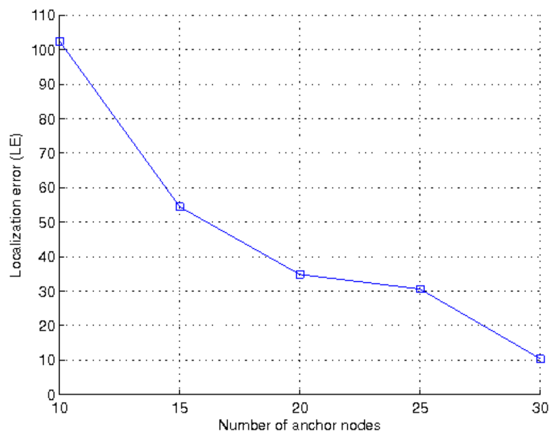


Figure 3.7: The trilateration and multilateration results with $r = 0.2$ and $nf = 10\%$.

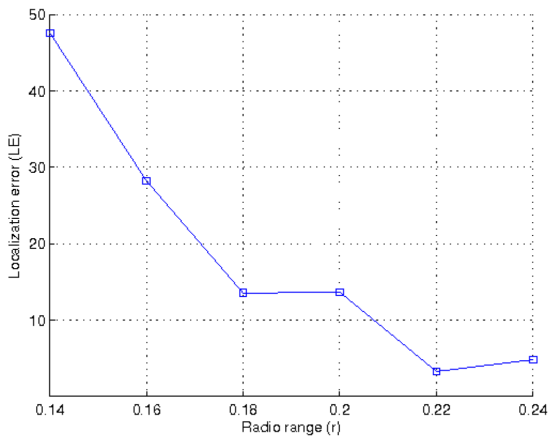
Table 3.9: Transmission range vs. average connectivity.

Trans. range	1.4	1.6	1.8	2.0	2.2	2.4
Connectivity	10	13	17	20	24	28

The summary of the computational experiences is that using the iterative multilateration technique with randomly selected four reference nodes, the localization error will be about 25%. This value can be decreased by increasing the anchor nodes number or the radio range. The found locations with the iterative multilateration can be also improved by solving the (3.36) general global optimization problem using the faster C-XSC solver.



(a) Number of anchor nodes vs. localization accuracy.



(b) Radio range vs. localization accuracy.

Figure 3.8: Impact of the number of anchor nodes and radio range on localization accuracy.

Chapter 4

Summary and conclusions

In this study, two important fields of the continuous global optimization have been considered: the stochastic and the interval arithmetic based global optimization. In both cases, we have implemented new algorithms in order to solve bound constrained global optimization problems. All implementations were done in MATLAB.

In the first part of the thesis we have presented the stochastic type method GLOBAL. We managed to improve the old algorithm in several aspects. The modified UNIRANDI local search method works now without dimension related problems. We use the BFGS local search algorithm instead of the earlier DFP method. We have also tested the new GLOBAL on standard test function and compared it with the old method and also with the C-GRASP method. On the basis of the testing we can state that the new implementation is at least as good or even better in terms of efficiency as the old one was, while the reliability of the solution has been increased substantially. Due to the better quasi-Newton local search method, the new version is much better for smooth problems even in terms of the necessary number of objective function evaluations.

We also evaluated the performance of the GLOBAL algorithm on the BBOB-2009 noiseless testbed, containing problems which reflect the typical difficulties arising in real-world applications. The results show that up to a small function evaluation budget, GLOBAL performs well.

The effectivity of the method have been illustrated by solving an optimization problem from the domain of pension system. The conclusion of the numerical test is that on the investigated problem the improved GLOBAL algorithm was able to find good approximations of the global minimizer points while the amount of computational efforts needed remained limited and in the acceptable region.

The second part of the thesis is concerned with interval arithmetic based branch and bound methods. Our aim was to provide an easy to use reliable global optimization method using MATLAB. We have proposed some algorithms which are based on the old method implemented in C-XSC. The first simple algorithm can solve global optimization problems without using derivative information. The second algorithm is a more sophisticated one using the common accelerating devices: the cutoff test, the concavity test, the monotonicity test, and the interval Newton step. Both algorithms were tested on standard test functions and compared with the old one. Summarizing the numerical results, we can state that the new implementation is in several indicators (e.g. number of function, gradient and Hessian evaluations, number of iterations, and memory complexity) in essence equivalent to the old one. The CPU time needed is as a rule by at least two order of magnitude higher for the INTLAB version – as it can be anticipated regarding the interpreter nature of MATLAB. In spite of the lower speed, the new interval global optimization methods can well be suggested as an early modeling and experimentation tool for the verified solution of bound constrained global optimization problems.

Furthermore, we described a new algorithm using a simple condition for applying the Newton step. We have completed a computational test in order to compare the new condition with the old one. Based on the results we can state that we managed to reduce significantly the necessary computation time and the total number of Hessian evaluations with the help of the new condition.

We also investigated the theoretical aspects of the Newton step on a predefined class of inclusion functions of the second derivative. More precisely, we analyzed those cases when the Newton step with the new condition is not successful in the sense that the width of the interval does not decrease or the current interval will be divided into many subintervals. According to the found results, we managed to characterize the cases when the Newton step is not successful.

Finally, we analyzed the localization problem of sensor networks using interval arithmetic techniques. We found an initial solution of sensor locations by iterative multilateration with randomly selected four reference nodes. The obtained localization error can be decreased by increasing the anchor nodes number or the radio range.

Összefoglalás

A doktori értekezés a folytonos globális optimalizálás két fontos témakörével, a sztochasztikus- valamint az intervallum aritmetikán alapuló módszerekkel foglalkozik. Mindkét módszer esetén a feltétel nélküli esetet vizsgáltuk és különböző algoritmusokat alkalmaztunk ezek megoldására. Valamennyi algoritmust MATLAB környezetben implementáltunk.

A dolgozat első részében a sztochasztikus módszerekkel foglalkoztunk. Bemutattuk az ismertebb módszereket, valamint azok elméleti hátterét. A dolgozat ezen részének alapja a GLOBAL két-fázisú, sztochasztikus optimalizáló módszer. Az algoritmust sikerült javítani a helyi kereső módszerek tekintetében. Az UNIRANDI helyi kereső eredményessége most már nem függ a feladat dimenziójától. A régi DFP kvázi-Newton alapú helyi kereső helyett, a hatékonyabb BFGS változatot használtuk. Az új GLOBAL algoritmust összehasonlítottuk a régi változattal, valamint egy szintén két-fázisú módszerrel, a C-GRASP-el. Az összevetések eredményeként megállapítható, hogy az új módszer legalább annyira jó a hatékonyság tekintetében, mint a régi GLOBAL módszer, a megbízhatóság alapján pedig egyértelmű a javulás a régi változathoz képest. A C-GARSP módszerrel való összehasonlítás eredményeként elmondható, hogy az új GLOBAL a függvényhívások számának tekintetében felülmúlja a másik módszert.

A GLOBAL módszert vizsgáltuk sima függvények halmazán, a BBOB-2009 konferencia keretén belül. A tesztfeladatok jól szemléltették a gyakorlati feladatok nehézségeit, és az eredmények alapján a GLOBAL nagyon jól teljesített kis függvényhívás számig.

Az eljárás hatékonyságát teszteltük egy valós feladaton is, amelyben optimális nyugdíjjárulék tervezés volt a cél. A numerikus tesztek alapján elmondható, hogy a GLOBAL jó közelítő megoldást talált az optimum helyre valamint a megfelelő futási idő is elfogadható.

A dolgozat második része intervallum aritmetikán alapuló korlátozás és szétválasztás típusú optimalizálási algoritmusok vizsgálatával foglalkozik. Egy numerikusan megbízható, könnyen használható, globális optimalizálási algoritmusnak az implementálása volt a cél, felhasználva a korábbi módszereket. Különböző algoritmus változatokat javaslunk a dolgozatban, amelyek struktúrája nagyjából követi a C-XSC-ben implementált régi eljárás szerkezetét.

Az első algoritmus célja olyan globális optimalizálási feladatok megoldása, amelyekben nem feltételezünk semmilyen információt a célfüggvény deriváltjáról. A második egy összetettebb algoritmus, amely már használja a következő gyorsító teszteket: kivágási-, konkavitási-, monotonitási teszt, és Newton lépés. Mindkét algoritmust numerikusan is teszteltük ismert függvényhalmazon és összehasonlítottuk a régi módszerrel. Összegezve a teszteredményeket elmondható, hogy az INTLAB alapú algoritmus hatékonysági mutatói nagyjából megegyeznek vagy hasonlók a C-XSC alapú eljárás mutatóival – a CPU idő kivételével. A CPU idő esetében egy-két nagyságrenddel nagyobb értékeket kaptunk, amely a MATLAB interpreter módban való működésének tudható be. Ennek ellenére az új globális optimalizálási módszer hasznos modellező eszköz lehet optimalizálási feladatok kezdeti tanulmányozására. Különösen igaz ez olyan feladatok esetén, amelyekre a CPU idő a növekedés ellenére is elfogadható mértékű.

Vizsgáltunk továbbá egy olyan algoritmust, amelyben a Newton lépés bekapcsolására egy új feltételt alkalmaztunk. Ennek az a lényege, hogy minden olyan intervallumra alkalmazzuk a Newton lépést, amelynek szélessége kisebb, mint egy előre megadott érték. Az INTLAB alapú algoritmust teszteltük az új feltétellel, és a kapott eredményeket összehasonlítottuk a korábbi eredményekkel. Ez alapján elmondható, hogy az új feltétel segítségével jelentősen sikerült csökkenteni a teljes futásidőt, valamint a Hesse mátrix kiértékelések számát.

A Newton lépés bekapcsolására használt új feltétel elméleti hátterét is megvizsgáltuk, a másodrendű derivált egy előre megadott befoglalófüggvény osztályán. Azokat az eseteket tanulmányoztuk, amikor az előbbi feltételt alkalmazva a Newton lépés nem eredményes abban az értelemben, hogy vagy nem is csökken az intervallum mérete, vagy sok darabra osztja fel az aktuális intervallumot. Az eredmények alapján elmondható, hogy sikerült jellemezni azokat az eseteket, amelyre a Newton lépés nem eredményes a fenti értelemben.

Befejezésként szenzorhálózatok lokalizálási problémáját vizsgáltuk intervallumos módszerekkel. Az iteratív ívmetszést alkalmazva kezdeti közelítő megoldásokat találtunk a szenzorok pozíciójára. Az iteráció során egy csomópont meghatározására négy, ismert pozíciójú szomszédos csomópontot választottunk véletlenszerűen. A lokalizálási hiba csökkenthető, ha az ismert helyzetű csomópontok számát, vagy a hatókör sugarát növeljük.

Appendix A

A collection of global optimization test problems

In the following, we list the objective functions denoted by f and the bounds used in our tests, the abbreviated and full names, the dimensionality and the known global minimizer points, and the corresponding function values of the problems.

S5: Shekel-5 ($x \in \mathbb{R}^4$):

$$f(x) = -\sum_{i=1}^{5} \frac{1}{(x - a_i)(x - a_i)^T + c_i}.$$

Coefficients:

i	a_i	c_i
1	(4.0, 4.0, 4.0, 4.0)	0.1
2	(1.0, 1.0, 1.0, 1.0)	0.2
3	(8.0, 8.0, 8.0, 8.0)	0.2
4	(6.0, 6.0, 6.0, 6.0)	0.4
5	(3.0, 7.0, 3.0, 7.0)	0.4

Bounds: $0 \leq x_i \leq 10, \ i = 1, \ldots, 4.$

Global minimum:

$$\begin{aligned} x^* &= (4.0000371, 4.0001332, 4.0000371, 4.0001332)^T, \\ f^* &= -10.15319967. \end{aligned}$$

S7: Shekel-7 ($x \in \mathbb{R}^4$):

$$f(x) = -\sum_{i=1}^{7} \frac{1}{(x - a_i)(x - a_i)^T + c_i}.$$

Coefficients:

i	a_i	c_i
1	(4.0, 4.0, 4.0, 4.0)	0.1
2	(1.0, 1.0, 1.0, 1.0)	0.2
3	(8.0, 8.0, 8.0, 8.0)	0.2
4	(6.0, 6.0, 6.0, 6.0)	0.4
5	(3.0, 7.0, 3.0, 7.0)	0.4
6	(2.0, 9.0, 2.0, 9.0)	0.6
7	(5.0, 5.0, 3.0, 3.0)	0.3

Bounds: $0 \leq x_i \leq 10,\ i = 1, \ldots, 4.$

Global minimum:

$$\begin{aligned} x^* &= (4.0005729, 4.0006893, 3.999489, 3.9996061)^T, \\ f^* &= -10.40294056. \end{aligned}$$

S10: Shekel-10 ($x \in \mathbb{R}^4$):

$$f(x) = -\sum_{i=1}^{10} \frac{1}{(x - a_i)(x - a_i)^T + c_i}.$$

Coefficients:

i	a_i	c_i
1	(4.0, 4.0, 4.0, 4.0)	0.1
2	(1.0, 1.0, 1.0, 1.0)	0.2
3	(8.0, 8.0, 8.0, 8.0)	0.2
4	(6.0, 6.0, 6.0, 6.0)	0.4
5	(3.0, 7.0, 3.0, 7.0)	0.4
6	(2.0, 9.0, 2.0, 9.0)	0.6
7	(5.0, 5.0, 3.0, 3.0)	0.3
8	(8.0, 1.0, 8.0, 1.0)	0.7
9	(6.0, 2.0, 6.0, 2.0)	0.5
10	(7.0, 3.6, 7.0, 3.6)	0.5

Bounds: $0 \leq x_i \leq 10,\ i = 1, \ldots, 4.$

Global minimum:

$$\begin{aligned} x^* &= (4.000746, 4.00059, 3.999663, 3.999509)^T, \\ f^* &= -10.53640981. \end{aligned}$$

H3: Hartman-3 ($x \in \mathbb{R}^3$):

$$f(x) = -\sum_{i=1}^{4} c_i \exp\left(-\sum_{j=1}^{3} a_{ij}(x_j - p_{ij})^2\right).$$

Coefficients:

i	a_i	p_i	c_i
1	(3.0, 10.0, 30.0)	(0.36890, 0.11700, 0.26730)	1.0
2	(0.1, 10.0, 35.0)	(0.46990, 0.43870, 0.74700)	1.2
3	(3.0, 10.0, 30.0)	(0.10910, 0.87320, 0.55470)	3.0
4	(0.1, 10.0, 35.0)	(0.03815, 0.57430, 0.88280)	3.2

Bounds: $0 \leq x_i \leq 1,\ i = 1, \ldots, 3.$

Global minimum:

$$\begin{aligned} x^* &= (0.1146143, 0.55564988, 0.85254695)^T, \\ f^* &= -3.86130579. \end{aligned}$$

H6: Hartman-6 ($x \in \mathbb{R}^6$):

$$f(x) = -\sum_{i=1}^{4} c_i \exp\left(-\sum_{j=1}^{6} a_{ij}(x_j - p_{ij})^2\right).$$

Coefficients:

i	a_i	p_i	c_i
1	(10.0, 3.0, 17.0, 3.5, 1.7, 8.0)	(0.1312, 0.1696, 0.5569, 0.0124, 0.8283, 0.5886)	1.0
2	(0.05, 10.0, 17.0, 0.1, 8.0, 14.0)	(0.2329, 0.4135, 0.8307, 0.3736, 0.1004, 0.9991)	1.2
3	(3.0, 3.5, 1.7, 10.0, 17.0, 8.0)	(0.2348, 0.1451, 0.3522, 0.2883, 0.3047, 0.6650)	3.0
4	(17.0, 8.0, 0.05, 10.0, 0.1, 14.0)	(0.4047, 0.8828, 0.8732, 0.5743, 0.1091, 0.0381)	3.2

Bounds: $0 \leq x_i \leq 1,\ i = 1, \ldots, 6.$

Global minimum:

$$\begin{aligned} x^* &= (0.2016895, 0.1500106, 0.4768739, 0.2753324, 0.31165161, 0.65730053)^T, \\ f^* &= -3.32236801. \end{aligned}$$

GP: Goldstein and Price ($x \in \mathbb{R}^2$):

$$\begin{aligned} f(x) &= (1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)) \cdot \\ &\quad (30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)). \end{aligned}$$

Bounds: $-2 \leq x_i \leq 2,\ i = 1, 2.$

Global minimum: $x^* = (0, -1)^T$, $f^* = 3$.

SHCB: Six-hump camel-back function ($x \in \mathbb{R}^2$):

$$f(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4.$$

Bounds: $-2 \leq x_i \leq 2,\ i = 1, 2.$

Global minimum:

$$\begin{aligned} X^* &= \left\{ \begin{array}{l} (0.08984201, -0.71265640)^T \\ (-0.08984201, 0.71265640)^T, \end{array} \right. \\ f^* &= -1.03162845. \end{aligned}$$

BR: Branin ($x \in \mathbb{R}^2$):

$$f(x) = \left(\frac{5}{\pi}x_1 - \frac{5.1}{4\pi^2}x_1^2 + x_2 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos x_1 + 10.$$

Bounds: $-5 \le x_1 \le 10$, $0 \le x_2 \le 15$.

Global minimum:

$$\begin{aligned} X^* &= \begin{cases} (-\pi, 12.275)^T \\ (\pi, 2.275)^T \\ (9.42478, 2.475)^T, \end{cases} \\ f^* &= 0.397887. \end{aligned}$$

RB2, RB5, RB10: Rosenbrock ($x \in \mathbb{R}^n$, $n = 2, 5, 10$, respectively):

$$f(x) = \sum_{i=1}^{n-1}\left[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right].$$

Bounds: $-1.2 \le x_i \le 1.2$, $i = 1, \ldots, n$.

Global minimum: $x^* = (\underbrace{1, \ldots, 1}_{n})^T$, $f^* = 0$.

ZH5, ZH10: Zakharov ($x \in \mathbb{R}^n$, $n = 5, 10$, respectively):

$$f(x) = \sum_{i=1}^{n} x_i^2 + \left(\sum_{i=1}^{n} 0.5ix_i\right)^2 + \left(\sum_{i=1}^{n} 0.5ix_i\right)^4.$$

Bounds: $-5 \le x_i \le 10$, $i = 1, \ldots, n$.

Global minimum: $x^* = (\underbrace{0, \ldots, 0}_{n})^T$, $f^* = 0$.

THCB: Three-hump camel-back function ($x \in \mathbb{R}^2$):

$$f(x) = 12x_1^2 - 6.3x_1^4 + x_1^6 + 6x_2(x_2 - x_1).$$

Bounds: $-3 \le x_i \le 3$, $i = 1, 2$.

Global minimum: $x^* = (0, 0)^T$, $f^* = 0$.

Easom: Easom ($x \in \mathbb{R}^2$):

$$f(x) = -\cos(x_1)\cos(x_2)\exp(-(x_1 - \pi)^2 - (x_2 - \pi)^2).$$

Bounds: $-100 \le x_i \le 100$, $i = 1, 2$.

Global minimum: $x^* = (\pi, \pi)^T$, $f^* = -1$.

Shubert: Shubert ($x \in \mathbb{R}^2$):

$$f(x) = \left(\sum_{i=1}^{5} i\cos((i+1)x_1 + i)\right)\left(\sum_{i=1}^{5} i\cos((i+1)x_2 + i)\right).$$

Bounds: $-10 \le x_i \le 10$, $i = 1, 2$.

Global minimum: 18 global minimizer points, $f^* = -186.7309$.

L3: Levy-3 ($x \in \mathbb{R}^2$):

$$f(x) = \sum_{i=1}^{5} i\cos((i-1)x_1 + i) \sum_{j=1}^{5} j\cos((j+1)x_2 + j).$$

Bounds: $-10 \le x_i \le 10,\ i = 1, 2.$

Global minimum:

$$X^* = \begin{cases} (4.97647760, 4.85805687)^T \\ (4.97647760, -1.42512842)^T \\ (4.97647760, -7.70831373)^T \\ (-1.30670770, 4.85805687)^T \\ (-1.30670770, -1.42512842)^T \\ (-1.30670770, -7.70831373)^T \\ (-7.58989301, 4.85805687)^T \\ (-7.58989301, -1.42512842)^T \\ (-7.58989301, -7.70831373)^T, \end{cases}$$
$$f^* = -176.54179313.$$

L5: Levy-5 ($x \in \mathbb{R}^2$):

$$\begin{aligned} f(x) &= \sum_{i=1}^{5} i\cos((i-1)x_1 + i) \sum_{j=1}^{5} j\cos((j+1)x_2 + j) \\ &\quad + (x_1 + 1.42513)^2 + (x_2 + 0.80032)^2. \end{aligned}$$

Bounds: $-10 \le x_i \le 10,\ i = 1, 2.$

Global minimum: $x^* = (-1.306853, -1.424845)^T$, $f^* = -176.137578.$

L8, L9, L10, L11, L12: Levy ($x \in \mathbb{R}^n$, $n = 3, 4, 5, 8, 10$, respectively):

$$\begin{aligned} f(x) &= \sum_{i=1}^{n-1} (y_i - 1)^2 (1 + 10\sin^2(\pi y_{i+1})) \\ &\quad + \sin^2(\pi y_1) + (y_n - 1)^2, \\ &\quad \text{with } \ y_i = 1 + (x_i - 1)/4,\ i = 1, \dots, n. \end{aligned}$$

Bounds: $-10 \le x_i \le 10,\ i = 1, \dots, n.$

Global minimum: $x^* = (\underbrace{1, \dots, 1}_{n})^T$, $f^* = 0.$

L13, L14, L15, L16, L18: Levy ($x \in \mathbb{R}^n$, $n = 2, 3, 4, 5, 7$, respectively):

$$\begin{aligned} f(x) &= \sum_{i=1}^{n-1} (x_i - 1)^2 (1 + \sin^2(3\pi x_{i+1})) \\ &\quad + (x_n - 1)^2 (1 + \sin^2(2\pi x_n)) + \sin^2(3\pi x_1). \end{aligned}$$

Bounds: $-10 \le x_i \le 10,\ i = 1, \dots, n$ for $n = 2, 3, 4$ and $-5 \le x_i \le 5,\ i = 1, \dots, n$ for $n = 5, 7$.

Global minimum: $x^* = (\underbrace{1, \dots, 1}_{n})^T$, $f^* = 0$.

Schw2.1: Beale ($x \in \mathbb{R}^2$):

$$f(x) = (1.5 - x_1 + x_1x_2)^2 + (2.25 - x_1 + x_1x_2^2)^2 + (2.625 - x_1 + x_1x_2^3)^2.$$

Bounds: $-1.5 \le x_1 \le 7.5,\ -4 \le x_2 \le 5$.

Global minimum: $x^* = (3, 0.5)^T$, $f^* = 0$.

Schw3.1: Schwefel ($x \in \mathbb{R}^3$):

$$f(x) = \sum_{i=1}^{3} \left((x_1 - x_i^2)^2 + (x_i - 1)^2 \right).$$

Bounds: $-10 \le x_i \le 10,\ i = 1, 2, 3$.

Global minimum: $x^* = (1, 1, 1)^T$, $f^* = 0$.

Schw2.5: Booth ($x \in \mathbb{R}^2$):

$$f(x) = (x_1 + 2x_1 - 7)^2 + (2x_1 + x_2 - 5)^2.$$

Bounds: $-5 \le x_i \le 5,\ i = 1, 2$.

Global minimum: $x^* = (1, 3)^T$, $f^* = 0$.

Schw2.7: Box 3D ($x \in \mathbb{R}^3$):

$$f(x) = \sum_{k=1}^{10} \left(\exp(\frac{-kx_1}{10}) - \exp(\frac{-kx_2}{10}) - (\exp(\frac{-k}{10}) - \exp(-k))x_3 \right)^2.$$

Bounds: $-10 \le x_i \le 10,\ i = 1, 2, 3$.

Global minimum: $x^* = (0, 0, 0)^T$, $f^* = 0$.

Schw2.10: Kowalik ($x \in \mathbb{R}^4$):

$$f(x) = \sum_{i=1}^{11} \left(a_i - x_1 \frac{b_i^2 + b_i x_2}{b_i^2 + b_i x_3 + x_4} \right).$$

Coefficients:

i	a_i	$\frac{1}{b_i}$
1	0.1957	0.25
2	0.1947	0.50
3	0.1735	1.00
4	0.1600	2.00
5	0.0844	4.00
6	0.0627	6.00
7	0.0456	8.00
8	0.0342	10.00
9	0.0323	12.00
10	0.0235	14.00
11	0.0246	16.00

Bounds: $0 \le x_i \le 0.42,\ i = 1, \dots, 4$.

Global minimum:

$$\begin{aligned} x^* &= (0.19283345, 0.19083623, 0.12311729, 0.13576598)^T, \\ f^* &= 3.074859878 \cdot 10^{-4}. \end{aligned}$$

Schw2.14: Powell ($x \in \mathbb{R}^4$):

$$f(x) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4.$$

Bounds: $-4 \le x_i \le 5,\ i = 1, \dots, 4$.

Global minimum: $x^* = (3, -1, 0, 1)^T$, $f^* = 0$.

Schw2.18: Matyas ($x \in \mathbb{R}^2$):

$$f(x) = 0.26(x_1^2 + x_2^2) - 0.48x_1x_2.$$

Bounds: $-30 \le x_i \le 30,\ i = 1, 2$.

Global minimum: $x^* = (0, 0)^T$, $f^* = 0$.

Schw3.2: Schwefel ($x \in \mathbb{R}^3$):

$$f(x) = \sum_{i=2}^{3} \left((x_1 - x_i^2)^2 + (x_i - 1)^2\right).$$

Bounds: $-1.89 \le x_i \le 1.89,\ i = 1, 2, 3$.

Global minimum: $x^* = (1, 1, 1)^T$, $f^* = 0$.

Schw3.7: Schwefel ($x \in \mathbb{R}^{10}$):

$$f(x) = \sum_{i=1}^{10} x_i^{10}.$$

Bounds: $-1.89 \le x_i \le 1.89,\ i = 1, \dots, 10$.

Global minimum: $x^* = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0)^T$, $f^* = 0$.

Griew5: Griewank-5 ($x \in \mathbb{R}^5$):

$$f(x) = \sum_{i=1}^{5} \frac{x_i^2}{400} - \prod_{i=1}^{5} \cos(\frac{x_i}{\sqrt{i}}) + 1.$$

Bounds: $-600 \leq x_i \leq 500,\ i = 1, \ldots, 5.$

Global minimum: $x^* = (0, 0, 0, 0, 0)^T,\ f^* = 0.$

Griew7: Griewank-7 ($x \in \mathbb{R}^7$):

$$f(x) = \sum_{i=1}^{7} \frac{x_i^2}{4000} - \prod_{i=1}^{7} \cos(\frac{x_i}{\sqrt{i}}) + 1.$$

Bounds: $-600 \leq x_i \leq 500,\ i = 1, \ldots, 7.$

Global minimum: $x^* = (0, 0, 0, 0, 0, 0, 0)^T,\ f^* = 0.$

R4: Ratz-4 ($x \in \mathbb{R}^2$):

$$f(x) = \sin(x_1^2 + 2x_2^2) \exp(-x_1^2 - x_2^2).$$

Bounds: $-3 \leq x_i \leq 3,\ i = 1, 2.$

Global minimum:

$$\begin{aligned} X^* &= \begin{cases} (0.0, -1.4575221047)^T \\ (0.0, 1.4575221047)^T, \end{cases} \\ f^* &= -0.10689134. \end{aligned}$$

R5, R6, R7, R8: Ratz ($x \in \mathbb{R}^n$, $n = 3, 5, 7, 9$, respectively):

$$f(x) = \left(\sin^2\left(\pi \frac{x_1 + 3}{4}\right) \sum_{i=1}^{n-1} \left(\frac{x_i - 1}{4}\right)^2 \left(1 + 20 \sin^2\left(\pi \frac{x_{i+1} + 3}{4}\right)\right)\right)^2.$$

Bounds: $-10 \leq x_i \leq 10,\ i = 1, \ldots, n.$

Global minimum: $x^* = (\underbrace{1, \ldots, 1}_{n-1}, [-10, 10])^T,\ f^* = 0.$

EX2: a simplified practical parameter estimation problem ($x \in \mathbb{R}^5$):

$$f(x) = \sum_{i=1}^{6} \left| f_i - \left(x_1 + \frac{x_2}{\omega_i^{x_3}} + \imath \left(\omega_i x_4 - \frac{x_5}{\omega_i^{x_3}} \right)\right)\right|^2,$$

where the f_i-s are $5.0 - 5.0\imath$, $3.0 - 2.0\imath$, $2.0 - \imath$, $1.5 - 0.5\imath$, $1.2 - 0.2\imath$ and $1.1 - 0.1\imath$, and $\omega_i = \pi i/20$, for $i = 1, 2, ..., 6$ (here $\imath$ is the imaginary unit, and i is an index).

Bounds: $[0.0, 1.0]^2 \times [1.1, 1.3] \times [0.0, 1.0]^2$.

Global minimum:

$$\begin{aligned} x^* &= (0.60629546, 0.55676269, 1.13180770, 0.75020138, 0.62190075)^T, \\ f^* &= 0.21245983. \end{aligned}$$

Bibliography

[1] G. Alefeld and J. Herzberger. *Introduction to Interval Computations.* Academic Press, New York, 1983.

[2] N. Baba. Convergence of a random optimization method for constrained optimization problems. *Journal of Optimization Theory and Applications*, 33:451–461, 1981.

[3] J. Balogh, T. Csendes, and T. Rapcsák. Some global optimization problems on Stiefel manifolds. *Journal of Global Optimization*, 30:91–101, 2004.

[4] J. Balogh, T. Csendes, and R.P. Stateva. Application of a stochastic method to the solution of the phase stability problem: cubic equations of state. *Fluid Phase Equilibria*, 212:257–267, 2003.

[5] J.R. Banga, C.G. Moles, and A.A. Alonso. Global optimization of bioprocesses using stochastic and hybrid methods. In C.A. Floudas and P.M. Pardalos, editors, *Frontiers in Global Optimization*, volume 74, pages 45–70. Springer-Verlag, Berlin, 2003.

[6] B. Bánhelyi, T. Csendes, and B.M. Garay. Optimization and the Miranda approach in detecting horseshoe-type chaos by computer. *International Journal of Bifurcation and Chaos*, 17:735–747, 2007.

[7] R.W. Becker and G.V. Lago. A global optimization algorithm. In *Proceedings of the 8th Allerton Conference on Circuits and System Theory*, pages 3–12, Monticello, Illinois, 1970.

[8] H.C.P. Berbee, C.G.E. Boender, A.H.G. Rinnooy Kan, C.L. Scheffer, R.L. Smith, and J. Telgen. Hit-and-run algorithms for the identification of nonredundant linear inequalities. *Mathematical Programming*, 37:184–207, 1987.

[9] M. Berz and G. Hoffstätter. Computation and application of Taylor polynomials with interval remainder bounds. *Reliable Computing*, 4:83–97, 1998.

[10] B. Betrò and F. Schoen. Sequential stopping rules for the multistart algorithm in global optimisation. *Mathematical Programming*, 38:271–286, 1987.

[11] B. Betrò and F. Schoen. Optimal and sub-optimal stopping rules for the Multistart algorithm in global optimization. *Mathematical Programming*, 57:445–458, 1992.

[12] C.J.P. Bélisle, H.E. Romeijn, and R.L. Smith. Hit-and-run algorithms for generating multivariate distributions. *Mathematics of Operations Research*, 18:255–266, 1993.

[13] P. Biswas and Y. Ye. Semidefinite programming for ad hoc wireless sensor network localization. In *Proceedings of the 3rd International Symposium on Information Processing in Sensor Networks*, pages 46–54, Berkeley, CA, USA, 2004.

[14] C.G.E. Boender, R.J. Caron, J.F. McDonald, A.H.G. Rinnooy Kan, H.E. Romeijn, R.L. Smith, J. Telgen, and A.C.F. Vorst. Shake-and-bake algorithms for generating uniform points on the boundary of bounded polyhedra. *Operations Research*, 39: 945–954, 1991.

[15] C.G.E. Boender and A.H.G. Rinnooy Kan. Bayesian stopping rules for a class of stochastic global optimization methods. Technical report, Econometric Institute, Rotterdam, 1985.

[16] C.G.E. Boender and A.H.G. Rinnooy Kan. Bayesian stopping rules for multistart global optimization methods. *Mathematical Programming*, 37:59–80, 1987.

[17] C.G.E Boender and A.H.G. Rinnooy Kan. On when to stop sampling for the maximum. *Journal of Global Optimization*, 1:331–340, 1991.

[18] C.G.E. Boender, A.H.G. Rinnooy Kan, G.T. Timmer, and L. Stougie. A stochastic method for global optimization. *Mathematical Programming*, 22:125–140, 1982.

[19] C.G.E. Boender and R. Zieliński. A sequential Bayesian approach to estimating the dimension of a multinominal distribution. In *Sequential Methods in Statistics. Banach Center Publications*, pages 37–42. PWN–Polish Scientific Publisher, 1982.

[20] S.H. Brooks. A discussion of random methods for seeking maxima. *Operations Research*, 6:244–251, 1958.

[21] C.G. Broyden. The convergence of a class of double-rank minimization algorithms. *Journal of the Institute of Mathematics and Its Applications*, 6:76–90, 1970.

[22] L.G. Casado and I. García. New load balancing criterion for parallel interval global optimization algorithm. In *Proceedings of the 16th IASTED International Conference*, pages 321–323, Garmisch-Partenkirchen, Germany, 1998.

[23] L.G. Casado, I. García, and T. Csendes. A new multisection technique in interval methods for global optimization. *Computing*, 65:263–269, 2000.

[24] L.G. Casado, I. García, and T. Csendes. A Heuristic Rejection Criterion in Internal Global Optimization Algorithms. *BIT*, 41:683–692, 2001.

[25] L.G. Casado, I. García, T. Csendes, and V.G. Ruiz. Heuristic rejection in interval global optimization. *Journal of Optimization Theory and Applications*, 118:27–43, 2003.

[26] A.E. Csallner, T. Csendes, and M.C. Markót. Multisection in interval branch-and-bound methods for global optimization–I. Theoretical results. *Journal of Global Optimization*, 16:371–392, 2000.

[27] T. Csendes. Nonlinear parameter estimation by global optimization – efficiency and reliability. *Acta Cybernetica*, 8:361–370, 1988.

[28] T. Csendes. New subinterval selection criteria for interval global optimization. *Journal of Global Optimization*, 19:307–327, 2001.

[29] T. Csendes. Numerical experiences with a new generalized subinterval selection criterion for interval global optimization. *Reliable Computing*, 9:109–125, 2003.

[30] T. Csendes, B. Bánhelyi, and L. Hatvani. Towards a computer-assisted proof for chaos in a forced damped pendulum equation. *Journal of Computational and Applied Mathematics*, 199:378–383, 2007.

[31] T. Csendes, B.M. Garay, and B. Bánhelyi. A verified optimization technique to locate chaotic regions of a Hénon system. *Journal of Global Optimization*, 35: 145–160, 2006.

[32] T. Csendes and L. Pál. A basic interval global optimization procedure for Matlab/INTLAB. In *Proceedings of International Symposium on Nonlinear Theory and its Applications (NOLTA2008)*, pages 592–595, Budapest, Hungary, 2008.

[33] T. Csendes, L. Pál, J.O.H. Sendín, and J.R. Banga. The GLOBAL optimization method revisited. *Optimization Letters*, 2:445–454, 2008.

[34] M.H. DeGroot. *Optimal statistical decisions.* McGraw-Hill, New York, 1970.

[35] L.P. Devroye. Progressive global random search of continuous functions. *Mathematical Programming*, 15:330–342, 1978.

[36] K.A. Dill, A.T. Phillips, and J.B. Rosen. Molecular structure prediction by global optimization. In I.M. Bomze, T. Csendes, R. Horst, and P.M. Pardalos, editors, *Developments in Global Optimization*, pages 217–234. Kluwer Academic Publishers, Dordrecht, 1997.

[37] L.C.W. Dixon. Global optima without convexity. Technical report, Numerical Optimization Center, Hatfield Polytechnic, Hatfield, England, 1978.

[38] P. Eső and A. Simonovits. Designing Optimal Benefit Rules for Flexible Retirement. Technical Report CMS-EMS 1353, Northwestern University, Evanston, 2002.

[39] P. Eső and A. Simonovits. Designing Optimal Benefit Rules for Flexible Retirement (in Hungarien: Optimális járadékfüggvény tervezése rugalmas nyugdíjrendszerre). *Közgazdasági Szemle*, 50:99–111, 2003.

[40] T.A. Feo and M.G.C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

[41] S. Finck, N. Hansen, R. Ros, and A. Auger. Real-parameter black-box optimization benchmarking 2009: Presentation of the noiseless functions. Technical Report 2009/20, Research Center PPE, 2009.

[42] P.E. Gill, W. Murray, and M.H. Wright. *Practical optimization.* Academic Press, New York, 1982.

[43] F. Glover and M. Laguna. *Tabu search.* Kluwer Academic Publishers, Dordrecht, 1997.

[44] A. Griewank and G. Corlis. *Automatic differentiation of algorithms: theory, implementation and applications.* SIAM, Philadelphia, 1991.

[45] C. Guss, C.G.E. Boender, and H.E. Romeijn. Stochastic methods. In R. Horst and P.M. Pardalos, editors, *Handbook of Global Optimization*, pages 829–869. Kluwer Academic Publishers, Dordrecht, 1995.

[46] R. Hammer, M. Hocks, U. Kulisch, and D. Ratz. *Numerical toolbox for verified computing.* Springer-Verlag, Berlin, 1993.

[47] E.R. Hansen. Global optimization using interval analysis: the one-dimensional case. *Journal of Optimization Theory and Applications*, 29:331–344, 1979.

[48] E.R. Hansen. Global optimization using interval analysis: the multi-dimensional case. *Numerische Mathematik*, 34:247–270, 1980.

[49] E.R. Hansen. *Global optimization using interval analysis.* Marcel Dekker, New York, 1992.

[50] N. Hansen, A. Auger, S. Finck, and R. Ros. Real-parameter black-box optimization benchmarking 2009: Experimental setup. Technical Report RR-6828, INRIA, 2009. URL HTTP://HAL.INRIA.FR/INRIA-00362649/EN/.

[51] N. Hansen, A. Auger, S. Finck, and R. Ros. Real-parameter black-box optimization benchmarking 2010: Experimental setup. Technical Report RR-7215, INRIA, 2010. URL HTTP://HAL.INRIA.FR/INRIA-00362649/EN/.

[52] N. Hansen, A. Auger, R. Ros, S. Finck, and P. Pošík. Comparing results of 31 algorithms from the black-box optimization benchmarking BBOB-2009. In *GECCO '10: Proceedings of the 12th annual conference comp on Genetic and evolutionary computation*, pages 1689–1696, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0073-5. doi: HTTP://DOI.ACM.ORG/10.1145/1830761.1830790.

[53] N. Hansen, S. Finck, R. Ros, and A. Auger. Real-parameter black-box optimization benchmarking 2009: Noiseless function definitions. Technical Report RR-6829, INRIA, 2009. URL HTTP://HAL.INRIA.FR/INRIA-00362633/EN/.

[54] E.R. Hansen and S. Sengupta. Bounding solutions of systems of equations using interval analysis. *BIT*, 21:203–211, 1981.

[55] M.J. Hirsch, C.N. Meneses, P.M. Pardalos, and M.G.C. Resende. Global optimization by continuous GRASP. *Optimization Letters*, 1:201–212, 2007.

[56] H.H. Hoos and T. Stützle. Evaluating Las Vegas algorithms—pitfalls and remedies. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 238–245, Morgan Kaufmann, San Francisco, CA, 1998.

[57] R. Horst and P.M. Pardalos. *Handbook of global optimization.* Kluwer Academic Publishers, Dordrecht, 1995.

[58] R. Horst and H. Tuy. *Global optimization: Deterministic approaches.* Springer-Verlag, Berlin, 1996.

[59] W. Huyer and A. Neumaier. Global optimization by multilevel coordinate search. *Journal of Global Optimization*, 14:331–355, 1999.

[60] T. Järvi. A random search optimizer with an application to a max-min problem. *Publications of the Institute for Applied Mathematics*, (3), University of Turku, 1973.

[61] A.A. Kannan, G. Mao, and B. Vucetic. Simulated annealing based wireless sensor network localization. *Journal of Computers*, 1:15–22, 2006.

[62] R.B. Kearfott. *Rigorous global search: continuous problems.* Kluwer Academic Publishers, Dordrecht, 1996.

[63] C.T. Kelley. *Iterative Methods for Optimization.* SIAM, Philadelphia, Pennsylvania, 1999.

[64] R. Krawczyk and K. Nickel. Centered form in interval arithmetics: quadratic convergence and inclusion isotonicity. *Computing*, 28:117–137, 1982.

[65] A.V. Levy and S. Gomez. The tunneling method applied for the global minimization of functions. In P.T. Boggs, editor, *Numerical Optimization*, pages 213–244. SIAM, Philadelphia, Pennsylvania, 1985.

[66] G. Mao, B. Fidan, and B.D.O. Anderson. Wireless sensor network localization techniques. *Computer Networks*, 51:2529–2553, 2007.

[67] M.C. Markót and T. Csendes. A New Verified Optimization Technique for the 'Packing Circles in a Unit Square' Problems. *SIAM Journal on Optimization*, 16: 193–219, 2005.

[68] M.C. Markót, J. Fernandez, L.G. Casado, and T. Csendes. New interval methods for constrained global optimization. *Mathematical Programming*, 106:287–318, 2006.

[69] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, Berlin, 1996.

[70] A. Mockus, L. Mockus, J. Mockus, W. Eddy, and G. Reklaitis. *Bayesian Heuristic Approach to Discrete and Global Optimization.* Kluwer Academic Publishers, Dordrecht, 1996.

[71] C.G. Moles, J.R. Banga, and K. Keller. Solving nonconvex climate control problems: pitfalls and algorithm performances. *Applied Soft Computing*, 5:35–44, 2004.

[72] C.G. Moles, G. Gutierrez, A.A. Alonso, and J.R. Banga. Integrated Process Design and Control Via Global Optimization – A Wastewater Treatment Plant Case Study. *Chemical Engineering Research and Design*, 81:507–517, 2003.

[73] M. Mongeau, H. Karsenty, V. Rouzé, and J.B. Hiriart-Urruty. Comparison of public-domain software for black box global optimization. *Optimization Methods and Software*, 13:203–226, 2000.

[74] R.E. Moore. *Interval analysis.* Prentice-Hall, Englewood Cliffs, New Jersey, 1966.

[75] J.A. Nelder and R. Mead. The downhill simplex method. *Computer Journal*, 7: 308–313, 1965.

[76] A. Neumaier. *Interval methods for systems of equations.* Cambridge University Press, 1990.

[77] A. Neumaier. Taylor forms – use and limits. *Reliable Computing*, 9:43–79, 2003.

[78] D. Niculescu and B. Nath. Ad Hoc Positioning System (APS). In *Proceeding of the Global Telecommunications Conference*, pages 2926–2931, San Antonio, CA, USA, 2001.

[79] E. Niewiadomska-Szynkiewicz and M. Marks. Optimization schemes for wireless sensor network localization. *International Journal of Applied Mathematics and Computer Science*, 19:291–302, 2009.

[80] I.H. Osman and J.P. Kelly. *Meta-Heuristics: Theory and Applications.* Kluwer Academic Publishers, Norwell, MA, USA, 1996.

[81] N.R. Patel, R.L. Smith, and Z.B. Zabinsky. Pure adaptive search in Monte Carlo optimization. *Mathematical Programming*, 43:317–328, 1989.

[82] L. Pál and T. Csendes. Improvements on the GLOBAL Optimization Algorithm with Numerical Tests. In *Proceedings of the 8th International Conference on Applied Informatics (ICAI2007)*, pages 101–109, Eger, Hungary, 2007.

[83] L. Pál and T. Csendes. INTLAB implementation of an interval global optimization algorithm. *Optimization Methods and Software*, 24:749–759, 2009.

[84] L. Pál and T. Csendes. An INTLAB based global optimization method and it's application to a sensor network localization problem (in Hungarian: Egy intervallum alapú globális optimalizálási módszer és alkalmazása szenzor lokalizálási feladatra). Közlésre beküldve, 2010.

[85] L. Pál and T. Csendes. Efficient estimation of loads in service networks. Accepted for publication, 2010.

[86] L. Pál, T. Csendes, M.C. Markót, and A. Neumaier. Black-box optimization benchmarking of the GLOBAL method. Submitted for publication, 2010.

[87] M. Piccioni and A. Ramponi. Stopping rules for the multistart method when different local minima have different function values. *Optimization*, 21:697–707, 1990.

[88] M. Pincus. A closed form solution of certain programming problems. *Operations Research*, 16:690–694, 1968.

[89] J. Pintér. Convergence properties of stochastic optimization procedures. *Optimization*, 15:405–427, 1984.

[90] J. Pintér. *Global optimization in action: continuous and Lipschitz optimization–algorithms, implementations, and applications.* Springer-Verlag, Berlin, 1996.

[91] P. Pošík, W. Huyer, and L. Pál. Benchmarking Several Global-Optimization Algorithms. Submitted for publication, 2010.

[92] M.J.D. Powell. How bad are the BFGS and DFP methods when the objective function is quadratic? *Mathematical Programming*, 34:34–47, 1986.

[93] M.J.D. Powell. The NEWUOA software for unconstrained optimization without derivatives. *Large-Scale Nonlinear Optimization*, 83:255–297, 2006.

[94] K. Price. Differential evolution vs. the functions of the second ICEO. In *Proceedings of the IEEE International Congress on Evolutionary Computation*, pages 153–157, Indianapolis, IN, USA, 1997.

[95] H. Ratschek and J. Rokne. *New computer methods for global optimization.* Ellis Horwood Chichester, UK, 1988.

[96] D. Ratz. *Automatische Ergebnisverifikation bei globalen Optimierungsproblemen.* PhD thesis, Univeristät Karlsruhe, 1992.

[97] D. Ratz. *Automatic slope computation and its application in nonsmooth global optimization.* Shaker-Verlag, Aachen, 1998.

[98] A.H.G. Rinnooy Kan and G.T. Timmer. Stochastic global optimization methods part I: Clustering methods. *Mathematical Programming*, 39:27–56, 1987.

[99] A.H.G. Rinnooy Kan and G.T. Timmer. Stochastic global optimization methods part II: Multi level methods. *Mathematical Programming*, 39:57–78, 1987.

[100] R.Y. Rubinstein. *Simulation and the Monte Carlo method.* Willey, New York, 1981.

[101] S.M. Rump. Algorithm with verified inclusions – theory and practice. In R.E. Moore, editor, *Reliability in Computing*, pages 109–126. Academic Press, 1988.

[102] S.M. Rump. INTLAB - INTerval LABoratory. In T. Csendes, editor, *Developments in Reliable Computing*, pages 77–104. Kluwer Academic Publishers, Dordrecht, 1999.

[103] C. Savarese, J. Rabaey, and K. Langendoen. Robust Positioning Algorithms for Distributed Ad-Hoc Wireless Sensor Networks. In *USENIX Annual Technical Conference*, pages 317–327. Monterey, CA, 2002.

[104] R. Schaefer. Foundations of Global Genetic Optimization. In J. Kacprzyk, editor, *Studies in Computational Intelligence Series*, volume 74. Springer-Verlag, 2007.

[105] J.O.H. Sendín, J.R. Banga, and T. Csendes. Extensions of a multistart clustering algorithm for constrained global optimization problems. *Industrial and Engineering Chemistry Research*, 48:3014–3023, 2009.

[106] Y. Shang, W. Rumi, Y. Zhang, and M. Fromherz. Localization from connectivity in sensor networks. *IEEE Transactions on Parallel and Distributed Systems*, 15: 961–974, 2004.

[107] S. Skelboe. Computation of Rational Interval Functions. *BIT*, 14:87–95, 1974.

[108] F.J. Solis and R.J.E. Wets. Minimization by random search techniques. *Mathematics of Operations Research*, 6:19–30, 1981.

[109] D. Stevenson. IEEE standard for binary floating point arithmetic (IEEE/ANSI 754-1985). Technical report, Floating-Point Working Group, Microprocessor Standards Subcommittee, 1985.

[110] P.G. Szabó, M.C. Markót, T. Csendes, E. Specht, and L.G. Casado. *New Approaches to Circle Packing in a Square: With Program Codes.* Springer-Verlag, Berlin, 2007.

[111] B. Tóth, J. Fernández, and T. Csendes. Empirical convergence speed of inclusion functions for facility location problems. *Journal of Computational and Applied Mathematics*, 199:384–389, 2007.

[112] A.A. Törn. Cluster analysis using seed points and density determined hyperspheres with an application to global optimization. In *Proceedings of the Third International Conference on Pattern Recognition*, pages 394–398, Coronado, California, 1976.

[113] A.A. Törn. A search clustering approach to global optimization. In L.C.W. Dixon and G.P. Szegő, editors, *Towards Global Optimization 2.* North-Holland, Amsterdam, The Netherlands, 1978.

[114] S. Voss, I.H. Osman, and C. Roucairol. *Meta-heuristics: Advances and trends in local search paradigms for optimization.* Kluwer Academic Publishers Norwell, MA, USA, 1999.

[115] Z. Yang and Y. Liu. Quality of Trilateration: Confidence-Based Iterative Localization. *IEEE Transactions on Parallel and Distributed Systems*, 21:631–640, 2010.

[116] Z.B. Zabinsky. *Stochastic Adaptive Search For Global Optimization.* Kluwer Academic Publishers, Dordrecht, 2003.

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

[118] Q. Zhang, J. Wang, C. Jin, J. Ye, C. Ma, and W. Zhang. Genetic algorithm based wireless sensor network localization. In *Fourth International Conference on Natural Computation, 2008. ICNC'08*, volume 1, pages 608–613, 2008.

[119] R. Zieliński. A stochastic estimate of the structure of multi-extremal problems. *Mathematical Programming*, 21:348–356, 1981.