KAUNAS UNIVERSITY OF TECHNOLOGY

Julius Žilinskas

BLACK BOX GLOBAL OPTIMIZATION: COVERING METHODS AND THEIR PARALLELIZATION

Doctoral dissertation

Technological sciences, informatics engineering (07T)

Kaunas, 2002

The research has been carried out at Kaunas university of technology during 1998-2002.

The license to conduct the doctoral studies was granted by the resolution No. 457 of the Government of Lithuania dated 14/04/1998.

The doctoral committee:

The main supervisor
Prof. Dr. Habil. (technological sciences, informatics engineering, 07T)
Rimantas ŠEINAUSKAS (Kaunas University of Technology);

committee members:
Prof. Dr. Habil. (technological sciences, informatics engineering, 07T)
Vytautas KAMINSKAS (Vytautas Magnus University),

Assoc. Prof. Dr. (technological sciences, informatics engineering, 07T)
Egidijus KAZANAVIČIUS (Kaunas University of Technology),

Prof. Dr. (technological sciences, informatics engineering, 07T)
Kaj MADSEN (Technical University of Denmark),

Prof. Dr. Habil. (physical sciences, informatics, 09P)
Vydūnas ŠALTENIS (Institute of Mathematics and Informatics).

Contents

Introduction

Importance of the problem

Many problems in engineering, physics, economic and other subjects are reduced to global minimization with many local minimizers. Mathematically the problem is formulated as

$$f^* = \min_{x \in D} f(x),$$

where $f(x)$ is a nonlinear function of continuos variables $f : \Re^n \to \Re$, $D \subseteq \Re^n$ is a feasible region, n is number of variables. Besides of global minimum f^* one or all global minimizers x^*: $f(x^*) = f^*$ should be found. No assumptions on unimodality are included into formulation of the problem.

The global optimization problems are classified difficult in the sense of the algorithmic complexity theory. Therefore global optimization algorithms are computationally intensive.

Large practical problems, unsolvable with available computers, always exist. When computing power of usual computers is not sufficient to solve a practical problem, the high performance parallel computers may be helpful. An algorithm is more applicable in case its parallel implementation is available, because larger practical problems may be solved by means of parallel computers. Therefore implementation and investigation of parallel versions of new algorithms is one part of the research.

Some subclasses of global optimization algorithms (e.g. random search, evolutionary strategies) are favorable to parallelization [68]. On the other hand, parallelization of adaptive covering techniques, of the methods based on statistical models and of some other methods is not straightforward. There is lack of research in this urgent subject. The aim of this research is to cover the gap related to covering methods.

Objectives and tasks of the research

The objectives of the research are:

- To develop and justify new versions of algorithms for black box global optimization inspired by covering techniques.
- To implement the proposed algorithms and to evaluate their performance and reliability by means of experimental testing.
- To implement the parallel versions of the proposed algorithms and to evaluate the performance of the implemented codes.
- To assess the applicability of the proposed algorithms by means of solving of typical practical problems.

To reach the objectives the following tasks should be fulfilled:

- The covering global optimization methods should be analyzed.
- The covering techniques should be modified to adopt them to a black box situation.
- The sequential versions of the proposed algorithms should be implemented.
- The performance of the implemented algorithms should be evaluated by means of experimental testing.
- The parallel computing techniques should be analyzed, especially the paradigms of communications between processors.
- The parallel algorithms for covering global optimization should be analyzed.
- The parallel versions of the proposed algorithms should be implemented and assessed with respect to criteria of parallelization.
- The applicability of implemented algorithms should be assessed by means of the results of optimization of typical practical problems.

Novelty and originality

- The covering global optimization methods have been modified increasing their performance and adopting them to a black box situation.
- New sequential and parallel algorithms have been implemented and evaluated.

Approval of the research

The results of the research are presented in the following conferences and seminars:

- International Workshop on Global Optimization 1999, University of Florence, Firenze, 1999.
- Conference “Informacinės technologijos’2000”, KTU, Kaunas, 2000.
- Second International Conference "Simulation, Gaming, Training, Business Process Reeingineering in Operations", RTU, Riga, 2000.
- Edinburgh University Seminar on Optimization, Edinburgh, 2001.
- TRACS-ACCESS-MINOS User Group Meeting, Edinburgh, 2001.
- BORNHOLM’02 Nordic Summer Course on Applied Optimization and Modeling, Bornholm, 2002.

Publications

The results of the research has been published in the following papers:

1. **O. Tingleff, A. Žilinskas, J. Žilinskas**. A Two Dimensional Optimization Algorithm Based on New Statistical Model of Multimodal Functions. *In: C. Carlsson et al. (eds.), Global & Multiple Criteria Optimization and Information System Quality,* Åbo Akademi Press, Finland, 103-112, 1998.
2. **J. Žilinskas**. Optimization of Lipschitzian functions by simplex based branch and bound. *Information Technology and Control No.1(14),* Kaunas, Technologija, 45-50, 2000.
3. **K. Madsen, J. Žilinskas**. Testing Real and Interval Methods for Global Optimization. *Technical report IMM-Report-2000-05,* Department of Mathematical Modelling, DTU, Denmark, 22 pages, 2000.
4. **K. Madsen, J. Žilinskas**. Evaluating Performance of Attraction Based Subdivision Methods for Global Optimization. *Second International Conference "Simulation, Gaming, Training, Business Process Reeingineering in Operations"*, RTU, Riga, 38-42, 2000.
5. **J. Žilinskas**. Black Box Global Optimization Inspired by Interval Methods. *Information Technology and Control No.4(21),* Kaunas, Technologija, 53-60, 2001.

Structure of the dissertation

The structure of the dissertation is as follows. In chapter 1 the global optimization methods are reviewed. The main attention is paid to the covering global optimization methods and the branch-and-bound technique to solve global optimization problems. In this chapter parallel computing is also analyzed. The load balancing and termination detection of parallel algorithms is analyzed in detail. The literature on the parallel branch-and-bound technique is reviewed. Criteria of efficiency of parallel algorithms are discussed.

In chapter 2 a new Lipschitz global optimization algorithm with simplicial partitioning is presented. The numerical results of experiments with the sequential algorithm are discussed and compared with results of other Lipschitz global optimization algorithms. The parallelization of algorithm is described. The parallel implementations are assessed with respect to criteria of parallelization.

In chapter 3 a new black box global optimization method inspired by interval arithmetic is presented. Sequential algorithm is implemented and investigated. The results of experimental testing are compared with the results of interval global optimization algorithm [43]. Parallel version of the algorithm is implemented.

In chapter 4 application of the presented black box global optimization method is presented. The method is applied to practical problems: multidimensional scaling, growth model of the human mandible and many body problems. The results of optimization are discussed.

The dissertation is completed with the conclusions and with the summary of the results.

1 Parallel computing from the perspective of global optimization

In this chapter the global optimization methods are reviewed. The main attention is paid to the covering global optimization methods and the branch-and-bound technique to solve global optimization problems. In this chapter parallel computing is also analyzed. The load balancing and termination detection of parallel algorithms is analyzed in detail. The literature on the parallel branch-and-bound technique is reviewed. Criteria of efficiency of parallel algorithms are discussed.

1.1 Global optimization

Mathematically the global optimization problem is formulated as

$$f^* = \min_{x \in D} f(x),$$

where a nonlinear function of continuos variables $f(x)$, $f : \Re^n \rightarrow \Re$, is called the *objective function*; $D \subseteq \Re^n$ is a *feasible region, n* is *number of variables*.

Besides of global minimum f^* one or all global minimizers x^*
$f(x^*)= f^*$
should be found. No assumptions on unimodality are included into formulation of the problem [35], [36], [80].

Sometimes an objective function is expressed analytically. However, the objective functions of practical problems usually are given by means of computer programs, and the properties of the objective function are difficult to elicit. The objective function values are assumed given by an *oracle* or a *black box*.

The global optimization problems are classified difficult in the sense of the algorithmic complexity theory. Therefore global optimization algorithms are computationally intensive, and solution time crucially depends on the dimensionality of a problem. In local optimization the decomposition approach is proved useful to reduce a problem of high dimensionality to several problems of lower dimensionality. The possibilities of decomposition of black box type objective functions are considered in [73].

1.1.1 Classification of global optimization methods

A classification of global optimization methods is proposed in [80]:

- Methods with guaranteed accuracy
  - Covering methods
- Direct methods
  - Random search methods
  - Clustering methods
  - Generalized descent methods
- Indirect methods
  - Methods approximating the level sets
  - Methods approximating the objective function

Random search methods may be adaptive and nonadaptive. Nonadaptive methods generate random trial points with predefined distribution. The trial points could be used as starting points of local searches. For example, pure random search does not contain local searches at all. Single start performs a single local search starting from the best trial point (with the smallest value of the objective function). Multi start performs local searches starting from all trial points and the global minimum is the smallest minimum found.

These methods are very simple but inefficient. Normally the probability of finding the global minimum approaches 1 when the number of observations of the objective function approaches infinity. Usually nonexperts use these methods to solve practical problems because the methods are simple and easy to implement. Sometimes these methods are used by the researchers to extract the characteristic of a problem: the global minimum, the number of global and local minimizers, probability that a local search started from a random point would reach the global minimum.

The parallelization of the nonadaptive methods is obvious. Each processor independently runs the same algorithm on equal parts of or on overall feasible region. There is no need to communicate between processors. The speedup is equal to the number of processors, the efficiency of parallelization is equal to 1.

The main idea of adaptive random search is to distribute the trial points nonuniformly in the feasible region with greater density in most promising subregions. The best function values found indicate the promising subregions. The practical efficiency of adaptive random search methods usually depends on the choice of rather many parameters and work well when used by their authors.

In the clustering methods the trial points are grouped into clusters identifying the neighborhoods of the local minimizers and just one local search is started from every cluster. The repeated descent to a local minimizer is prevented. The trial points may be sampled using a grid or randomly.

The generalized descent methods are the generalization of the local search methods to global optimization. In the trajectory methods the differential equation describing the local descent is modified. In the penalty methods the local search algorithm is repeatedly applied to a modified objective function preventing the descent to known local minima.

In the methods approximating the objective function the statistical models of the objective function are used. The unknown values of the objective function are modeled using random variables. The auxiliary computations to determine the next trial point are expensive, therefore these methods are reasonable for the expensive objective functions. The optimization technique based on a stochastic function model and minimization of the expected deviation of the estimate from the real global optimum is called Bayesian [57].

Theoretically the covering methods solve the global optimization problems of some classes with guaranteed accuracy. Covering methods detect the subregions not containing the global minimum and discard them from further search. The partitioning of the subregions stops when global minimizers are bracketed in small multidimensional intervals guaranteeing the prescribed accuracy.

A lower bound of the objective function over the subregion may be used to indicate the subregions who can be discarded. Some methods are based on lower bound constructed as convex envelope of an objective function [27], [38], [39]. Lipschitz optimization assumes that the slope of an objective function is bounded. Interval methods estimate the range of a function over an interval using interval arithmetic. A branch and bound technique can be used for managing the list of subregions and the process of discarding and partitioning.

Covering methods are usually not applicable in a black box situation. In our research we are interested in development of covering methods, improving their performance and making them applicable in a black box situation.

1.1.2 Lipschitz methods

Lipschitz optimization is based on the assumption that the slope of an objective function is bounded [36], [61]. A function $f : D \rightarrow \Re$, $D \subseteq \Re^n$ is said to be Lipschitz if it satisfies the condition

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

where L>0 is a constant called Lipschitz constant, D is compact and $\|\cdot\|$ denotes the Euclidean norm; other norms could be also considered.

In [31] several Lipschitz global optimization methods are discussed and compared experimentally, the advantages and disadvantages of this approach are discussed. Lipschitz global optimization is applicable when Lipschitz constant of the objective function is available. The efficient algorithms exist for the one-dimensional case. In this case the lower bound is inexpensive to compute. However, the calculation of tight lower bound becomes expensive in multidimensional case. Moreover, the Lipschitz constant depends on scaling of the variables and optimization is inefficient when scaling is made not properly. If an estimate of the Lipschitz constant is too small then the global minimizer may be missed. On the other hand overestimation of the Lipschtiz constant implies the overestimated bounds and loss of the optimization efficiency. Another difficulty is that it is not taken into account that the Lipschitz constant may vary greatly over different regions of the domain.

The Lipschitz constant normally is unknown for practical problems whose objective functions are defined by means of computer code. In [84] a method for estimating the

Lipschitz constant is presented. Extensions of Lipschitz optimization for the case of unknown Lipschitz constant are considered in [62].

1.1.3 Interval methods

Interval global optimization methods are based on interval arithmetic proposed in [58]. The interval arithmetic operates with real intervals $X = [\underline{x}, \overline{x}] = \{x \in \Re \mid \underline{x} \le x \le \overline{x}\}$, where $\underline{x}$ and $\overline{x}$ are real numbers. Any real arithmetic operation *x* op *y* has the corresponding interval arithmetic operation *X* op *Y*:

$$X \operatorname{op} Y = [\min_{x \in X, y \in Y} x \operatorname{op} y, \max_{x \in X, y \in Y} x \operatorname{op} y] = \{x \operatorname{op} y \mid x \in X, y \in Y\}.$$

X op *Y* contains every possible number which can be formed by *x* op *y* for each $x \in X$ and $y \in Y$. The common interval arithmetic operations are defined as:

$$X + Y = [\underline{x} + \underline{y}, \overline{x} + \overline{y}],$$

$$X - Y = [\underline{x} - \overline{y}, \overline{x} - \underline{y}],$$

$$X \cdot Y = [\min(\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y}), \max(\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y})],$$

$$X / Y = [\underline{x}, \overline{x}] \cdot [1/\overline{y}, 1/\underline{y}] \text{ if } 0 \notin Y.$$

Let a function *f*(*x*) be defined over the "box" $X = \{x \in \Re^n \mid \underline{x}_i \le x_i \le \overline{x}_i\}$. The lower and upper bounds for the function values can be estimated applying the interval operations with the intervals $[\underline{x}_i, \overline{x}_i]$ instead of the real operations with x_i in the algorithm of calculation the function values. The bounds are useful to detect the subregions of the feasible region not containing a global minimizer. Such subregions may be discarded from the further search. If the objective function is differentiable it is possible to compute the intervals of the derivatives and discard the subregions where the objective function is monotone. If the objective function is twice continuously differentiable it is possible to compute the intervals of the second derivatives and discard the subregions where the objective function is concave. If the objective function is twice differentiable the special interval Newton method can be applied to reduce the subregions, and discard the subregions where there are no stationary points [30].

The disadvantage of interval methods is that they need mathematical expression of the objective function, and they are not applicable in the black box situation. Another disadvantage is the dependency problem [30]: when a given variable occurs more than once in an interval computation, it is treated as a different variable in each occurrence. The latter causes widening of computed intervals making it more difficult to obtain tight intervals. One should always be aware of this shortcoming and take appropriate steps to reduce its effect. However it is not always possible to overcome the problem, when the objective function is defined by means of a computer code.

The first version of the interval global optimization algorithm was oriented to minimization of a rational function by bisection of sub-domains [69]. Interval methods for global optimization were further developed in [28], [29], [59], e.g. the interval Newton method and the test of *strict monotonicity* were introduced. A thorough description including theoretical as well as practical aspects can be found in [30] where the very efficient interval global optimization method involving monotonicity and nonconvexity tests and special interval Newton method is proposed. The method assumes that the objective function is twice continuously differentiable. The mathematical expressions of the functions should be available. If the monotonicity and nonconvexity tests and interval Newton method are not used the method can minimize even noncontinuous functions, but then it is not so efficient.

The interval methods has been combined with searches implemented in real number arithmetic, see for instance [7], [40]. In [43] and [40] the global unconstrained minimization method involving a combination of local search, branch-and-bound technique and interval arithmetic is proposed. In this method derivatives are not required.

1.1.4 Branch and bound technique

Branch and bound is a technique to solve combinatorial and covering global optimization problems. An iteration of a classical sequential branch and bound algorithm processes a node in the search tree representing a not yet explored subspace of the solution space [8]. Iteration has three main components: selection of the node to process, bound calculation and branching. Partitions obtained with branch and bound algorithms for global optimization differ from those used in combinatorial optimization in that [31]:

- The number of possible partitions is infinite.
- The classes of partitions may overlap.

Let *LB*(*X*) and *UB*(*X*) denote upper and lower bounds of $f_{\min}$ over *X*: $LB(X) \leq \min_{x \in X} f(x)$, $UB(X) \geq \min_{x \in X} f(x)$. The general branch and bound algorithm is shown in Algorithm 1.

Algorithm 1. General branch and bound algorithm.

```
$UB(X) \Leftarrow \infty$.
Cover the feasible region X by $C = \{C_j \mid j = 1, \ldots, m\}$, $X \subseteq \cup C_j$.
While $C \neq \varnothing$,
        Select $B \in C$, $C \Leftarrow C \setminus \{B\}$.
        Branch B: $B \subseteq \bigcup_{j=1}^{p} T_j$, $\mathrm{T} = \{T_j \mid T_j \cap X \neq \varnothing\}$.
        $UB(X) \Leftarrow \min(\{UB(X)\} \cup \{UB(T_j \cap X) \mid j = 1, \ldots, p\})$.
        $C \Leftarrow \{B : C \cup T \mid LB(B) < UB(X) + \varepsilon\}$.
```

C is called the *candidate set*. The branch and bound scheme aims to reduce *C* and make it converge to X^*.

The rules of covering, selection, branching and bounding differ from algorithm to algorithm.

The rules of covering and branching depend on type of partitions used. Partitions may be hyper-rectangular, simplicial, hyper-conic or hyper-spherical. Usually feasible regions of general global optimization problems are hyper-rectangles. All interval and most of Lipschitz global optimization branch and bound algorithms use the hyper-rectangular partitions. In this case initial covering is simple: *C*={*X*}. Covering by hyper-spheres causes overcovering of feasible region as well as overlapping of spheres themselves. In [12] a two dimensional case is considered where branching is implemented as partitioning into regular sub-simplices with initial over-covering. Such a non-overlapping branching is not known in more than two dimensions. The use of irregular simplices enables nonovercovering of feasible region as well as nonoverlapping branching.

There are three main strategies of selection:

- Best first – select an element of *C* with minimal lower bound.
- Depth first – select the youngest element of *C*.
- Breadth first – select the oldest element of *C*.

The bounding rule describes how the bounds of minimum are found. For the upper bound the best currently found value of the objective function might be accepted. The lower bound may be estimated using convex envelopes of function values, using the Lipschitz condition or interval arithmetic.

1.2 Parallel computers

When computing power of usual computers is not sufficient to solve a practical problem, the high performance parallel computers may be helpful. Parallel computers are classified to be a shared memory multiprocessor system or message-passing multicomputer system.

The shared memory multiprocessor system consists of multiple internal processors and the shared memory, as it is shown in Figure 1. The shared memory has a single address space and each processor can access it. The data is stored in the shared memory. The processors communicate through shared data structures what makes parallel programming attractive.

The shared memory can become a bottleneck when processors frequently access the shared memory. It is difficult to implement the hardware to achieve fast access to all the shared memory by all the processors, especially when there are many processors. The hierarchical memory structure is usually implemented involving the fast local cache memory and the mechanism of updating the global memory (cache coherency).

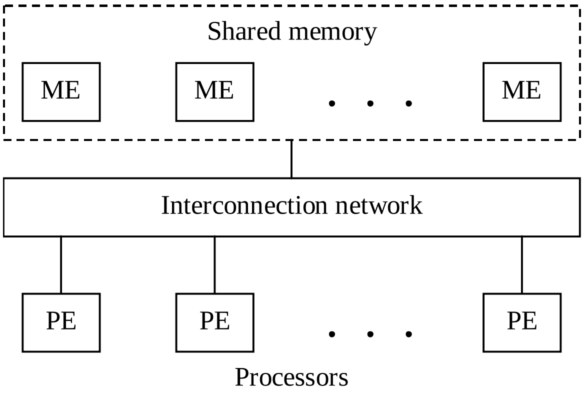


Figure 1. Shared memory multiprocessor system.

In a distributed message-passing multicomputer system each processor has its local memory which is not accessible by other processors. The memory is distributed. The interconnection network should be provided for processors to communicate. Special mechanisms for controlling simultaneous access of data are not necessary. A distributed message-passing multicomputer system is shown in Figure 2.

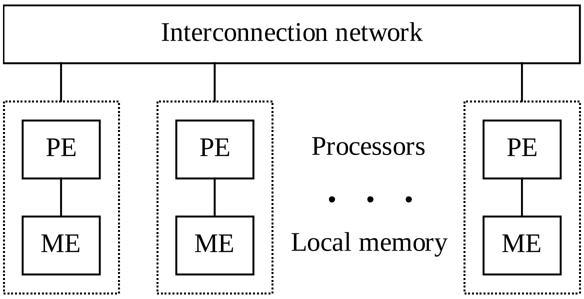


Figure 2. Distributed message-passing multicomputer system.

Such a system may be a specially designed computer system or usual computers connected using an interconnection network. Specially designed systems usually have static interconnection networks [82] who have direct physical links between nodes. Possible topologies of the static network are [20], [70], [82] completely connected network, line-mesh, ring-torus, star, tree, hypercube.

Processors communicate using messages who include data. The message-passing multicomputer will physically *scale* easier than a shared memory multiprocessor, what means it can more easily be made larger [82].

Implementation of parallel programs for message-passing system is more complicated than for shared memory systems [82]. However, the message-passing paradigm is more general. It can be implemented in a shared memory multiprocessor by using shared memory to hold data to be sent between processors. Therefore it is possible to implement parallel message-passing algorithms portable among both types of parallel systems – shared memory and message-passing multiprocessors. There exists a standardized portable communication protocol used on massively parallel machines, the Message Passing Interface (MPI) [55]. Because of its generality the message-passing model has been chosen in our research for implementation of parallel programs.

INMOS has introduced transputer – a multicomputer building block. Transputer is a specially designed device, which integrates a reduced-instruction-set-computer processor, some memory and a set of inter-processor communication links [83], [71]. Some families of transputers have been manufactured: T222, T425, T800. Multiprocessor boards based on transputers for personal computers were relatively low cost parallel-processing solutions. The transputer network topology of such boards may be set up as a pipeline, a ring, a binary tree or a cube [76]. The Occam language [37] developed by INMOS was used to implement parallel programs on transputer systems.

Parallel computing is one of the most developing branches of science and technology. The top list of most powerful supercomputers is updated twice a year [78]. There is number of international journals and conferences on parallel computing. There was lack of parallel computer systems in Lithuania. Therefore parallel computing was not very popular here. However it is developing. There are IBM RS/6000 SP parallel computer with four processors at Vilnius Gediminas Technical University and some clusters of workstations. A book on parallel algorithms for students of mathematics and computer science [13] is published at Vilnius Gediminas Technical University. Few doctoral dissertations related to parallel computing are defended, for example [63], [72].

There are some published results on parallel computing research in Lithuania. In [15] the implementation of numerical adaptive algorithms for multi-dimensional quadrature on distributed-memory parallel systems is described. The algorithms are targeted at clusters of workstations with standard message passing interfaces. The most important issues are communication and load balancing. Numerical results on various workstation clusters are reported.

Parallel numerical integration algorithms for multi-dimensional integrals are considered in [14]. A new hyper-rectangle selection strategy is proposed for the implementation of globally adaptive parallel quadrature algorithms. The master-slave parallel algorithm prototype is used for the realization of the algorithm. Numerical results on the SP2 computer and on a cluster of workstations are reported.

The analysis of the method for multiple criteria optimization problems applying a computer network has been proposed in [19]. The essence of the proposed method is the distribution of the concrete optimization problem into the network rather than the parallelization of some optimization method.

1.3 Load balancing and termination detection

The development of on algorithm in order to run the parts of the algorithm in parallel is called *parallelization*. The parallelization aims to obtain the smallest possible time of execution of the algorithm. The processors should not become idle, i.e. not contributing to common goal. Ideally the processors should compute continuously and finish together. The possibly uniform distribution of the work between processors is called *load balancing*. The detection when the computations are completed is called *termination detection*.

1.3.1 Centralized load balancing

The load balancing is called centralized if one processor dynamically distributes the tasks and the other processors do not interchange them between each other. This corresponds to the master-slave paradigm. The master processor holds the collection of tasks, so called *work pool* or *task queue*. When the slave processor becomes idle it requests a task from the master processor. The master processor sends out the most complex or the most important task it has. When the slave finishes the task it sends the results to the master processor.

The master processor also manages the termination. In this case it is not difficult to detect the termination conditions defined as follows:

- The task queue is empty.
- All slave processors request a task.

The second condition means that all slaves have finished their tasks and have sent the results to master. If a slave processor is still running it is possible that new tasks will be generated. The disadvantage of this scheme is that the master can become a bottleneck especially when the system is large and the communications are frequent: the slaves can become idle when waiting for new tasks.

1.3.2 Static load balancing

The load balancing is called static if the work is distributed initially before the computations start. The synonyms of the static load balancing are static mapping, static scheduling. When the number of initial tasks is equal to the number of processors each processor gets one task. When the number of tasks is larger the static load balancing techniques are used to distribute the tasks among the processors:

- Round robin algorithm distributes tasks in sequential order of processors coming back to the first processor after the last one.
- Randomized algorithms distribute tasks in random order.
- Optimization techniques distribute tasks minimizing the time of execution of the parallel algorithm.

The optimization techniques could be used for the static load balancing when the execution time for each task is known in advance. However it is very difficult to estimate the execution times without actually executing the program. Moreover the system may be heterogeneous – processors and interconnections may be of different type and speed.

The termination is easy because the processors do not communicate the tasks. When the processors finish their work they terminate. However the load balance may be poor since the processors may finish their work at different times.

1.3.3 Distributed dynamic load balancing

By distributed dynamic load balancing the processors exchange tasks between themselves. The aim is to share the work uniformly. Processors perform calculations as well as communicate to exchange the tasks.

The transfer of task could be initiated either by the receiver or by the sender. When the processor has few or no tasks it initiates the transfer: it requests a task from another processor, which sends one or more tasks if it has. An overloaded processor may initiate the transfer: it sends one or more tasks to the other processors it selects. However, the processor should determine that it is overloaded, what means that it has more tasks than the others. The determination could be expensive, because the processors should know the load of other processors.

Two dynamic distribution strategies are indicated in [11]:

- “on demand” is the receiver initiated strategy,
- “on overload” is the sender initiated strategy.

A local approach of determination overload is presented in [11].

The processor could be selected for communication having the structure of the interconnection network in mind. If the interconnection network is known, the nearest neighbors are usually selected. If the structure is general or unknown, all processors are equal candidates and any other processor could be selected. Each processor has its own local selection algorithm. The processor could be selected using round robin or random algorithms. The round robin algorithm selects candidate processor in sequential order coming back to the first processor after the last one. The random selection algorithm selects the random processor from the candidate set.

Five dynamic load balance strategies are discussed in [81]:

- Sender Initiated Diffusion (SID) is an asynchronous highly distributed local approach using near-neighbor information only. The heavily loaded sender processor initiates balancing.
- Receiver Initiated Diffusion (RID) is similar to SID approach. The underloaded receiver processor initiates balancing.
- Hierarchical Balancing Method (HBM) is an asynchronous global approach which organizes the system into hierarchy of subsystems and balances load at different levels of hierarchy.
- Gradient Method (GM) uses a gradient map of the proximities of underloaded processors in the system to guide the migration of tasks between overloaded and underloaded processors.
- Dimension Exchange Method (DEM) is a global, fully synchronous, iterative approach.

The strategies illustrate the tradeoff between:

- Knowledge – the accuracy of balancing decision,
- Overhead – the amount of added processing and communication incurred by the balancing.

The results of experiments with implementations of all five strategies on an Intel iPSC/2 hypercube indicate that the RID approach performs well and can be most easily be scaled to support highly parallel systems [81].

Using the distributed dynamic load balancing the work could be distributed uniformly. However communications between processors cause an overhead during execution, because some time is spent on communications. Sometimes there could be not enough tasks to share.

1.3.4 Termination detection

To detect the termination of a distributed computation may be difficult. The general distributed termination conditions are:

- All the processors have finished their tasks. This is local termination condition.
- There are no tasks in transit. The task in transit may restart a terminated processor.

It is not difficult to detect the first condition. However the second condition could not be detected locally. One of the termination techniques should be used. The distributed termination method using request and acknowledgement messages is general method to cope with tasks in transit [82]. In [4] the formal mathematical arguments of the method are given in detail. In this method each processor is in one of the states – active or inactive. Initially a processor has no tasks and is inactive. It becomes active when it gets a task from other processor. The activating processor becomes the parent of the activated processor. In this way a tree of active processors are created. Each active processor has a unique parent. The active processor may receive more tasks from other processors. If it gets the task from the processor who is not his parent, it has to send an acknowledgement message. The processor sends acknowledgement message to its parent when it becomes inactive. It becomes inactive when:

- It has finished all its tasks. The local termination condition is satisfied.
- It has sent all its acknowledgement messages for the tasks it has received.
- It has received all the acknowledgment messages for the tasks it has sent.

The processor becomes inactive before its parent processor. When the root processor becomes inactive the computation can terminate.

Some other termination algorithms are given in [82]. The ring termination algorithms organize processors in a ring structure and use tokens to detect termination conditions. The fixed energy distributed termination algorithm uses notation of a fixed quantity within the system. The energy is spread out when distributing the tasks and it is returned after the tasks are completed. The termination conditions are satisfied when all the energy is returned to the root and the root finishes its tasks.

The termination method using request and acknowledgement method was chosen for distributed algorithms with dynamic load balancing because of its generality and its proven soundness [82].

1.4 Parallel branch-and-bound

The literature on parallel branch-and-bound published up till 1994 has been reviewed in [22], where a new classification of parallel branch and bound algorithms has been proposed. The authors identify three main approaches in designing parallel branch-and-bound algorithms:

- *Parallelism of type* 1 introduces parallelism performing the operations on generated subproblems. It consists, for example, of executing the bounding operation in parallel for each subproblem to accelerate the execution. Thus, this type of parallelism has no influence on the general structure of the branch-and-bound algorithm and is particular to the problem. For example, the objective function may be computed in parallel.
- *Parallelism of type* 2 consists of building the branch-and-bound search tree in parallel by performing operations on several subproblems simultaneously. Hence, this type of parallelism may affect the design of the algorithm.
- *Parallelism of type* 3 implies that several search branch-and-bound trees are built in parallel. The trees are characterized by different operations (branching, bounding, testing for elimination, or selection), and the information generated when building one tree can be used for the construction of another.

The second type of parallelism may be further classified according to the implementation techniques [22]. *Synchronous* algorithms are executed in phases. In each phase processors work independently and communicates only between phases. Processors must synchronize before communications what cause overhead. In *asynchronous* algorithms processors may communicate at any unpredictable time. These algorithms have nondeterministic behavior.

The second implementation difference depends on the location of the *work pool*, i.e. the store of subproblems. *Single* pool algorithms store the subproblems in one memory location. Single pool algorithms may be implemented using the master-slave paradigm presented in Section 1.3.1. In this case the master processor manages the single work pool. *Multiple* pool algorithms use several work pools – store the subproblems in several memory locations. These algorithms correspond to distributed paradigm.

According to the classification of [22] parallel branch-and-bound algorithms of type 2 are classified as follows:

- *Synchronous Single Pool* (SSP),
- *Asynchronous Single Pool* (ASP),
- *Synchronous Multiple Pool* (SMP),
- *Asynchronous Multiple Pool* (AMP).

General features of known branch-and-bound algorithms depending to the different subclasses are analyzed in [22]. Authors conclude that synchronization is unnecessary in most cases, ASP algorithms are appropriate only for problems with a nontrivial bounding operation and parallel architectures with small number of processors.

A slightly different classification of parallel branch-and-bound algorithms is proposed in [54]. The algorithms are characterized by the implementation of the work pool and synchronization:

- *Select Highest Overall* (SHO). Highly synchronized algorithms with a global work pool.
- *Select Highest Available* (SHA). Algorithms with a global work pool but without synchronization.
- *Select Highest Locally* (SHL). Each unsynchronized processor has its own work pool.
- *Select Highest with a shared pool* (SHO_n and SHA_n) are based on SHO and SHA respectively. Processors are grouped into *n* groups (*n* is less than the number of processors) and all processors in the same group share the same work pool. SHO_1 and SHO, SHA_1 and SHA are equivalents respectively.

The results of the experiments with SHO, SHA and SHL algorithms are given in [54]. The results show that SHA performs best when the evaluation of the subproblem is reasonably accurate and cost time, and there are a small number of processors. The SHL performs best for problems with non-accurate and cheap evaluation of the subproblems. When the time to evaluate the subproblem is not constant, the synchronization step is costly and SHO performs badly.

Parallel branch-and-bound algorithms of different classes may be combined. A parallel branch-and-bound algorithm with two phases: a synchronous initialization and an asynchronous exploration, is proposed in [23]. The synchronous initialization phase is a generalization of the sequential best-first search strategy. In the exploration phase the asynchronous processors perform independent depth-first searches.

An original depth-first branch-and-bound algorithm for shared memory machines is proposed in [50]. The feeding tree is introduced. It allows a good distribution of tasks to the processors. The shared feeding tree is the upper part of the search tree developed down to the predefined depth. The leaves of the feeding tree are the roots of the subtrees allocated to the processors. The algorithm was executed on shared memory machine on up to 4 processors. The results show that speedup is nearly equal to the number of processors.

The search strategy (the selection criterion) in parallel branch-and-bound is discussed in [10]. The best first and the depth first search strategies are tested on the Quadratic Assignment and the Job Scheduling combinatorial optimization problems. Strategies are compared in terms of running times and in terms of number of performed bound calculations.

Parallel branch-and-bound algorithms may be classified by means of the used load balancing strategy:

- Centralized load balancing,
- Static load balancing,
- Distributed dynamic load balancing.

Similar classification was used in [8]:

- The Master-Slave Paradigm,
- Distributed Branch-and-Bound,
- Parallel Branch-and-Bound with static work distribution.

Parallel branch-and-bound algorithms may correspond to one of two main paradigms of parallel programming: master-slave or distributed.

In the parallel master-slave system there is one master processor, which controls the optimization process. The slave processors receive subproblems from the master, perform computations, and send the results to the master. It is ensured that all slaves perform computations on promising subproblems, the master controls load balancing and termination. The disadvantage of this scheme is the potential bottleneck of the master especially when the system is large and the communications are frequent and long lasting. The slaves may become idle when waiting for subproblems.

The second paradigm, i.e. distributed parallelization, assumes a cooperative work of a set of equal communicating processors. The main problems of the distributed parallelization are load balancing and termination detection.

There are two different balancing strategies: static and dynamic. When the static load balancing is used, the subproblems are initially distributed and then the processors work independently and do not exchange any later generated subproblems. The application of static load balancing to the considered problem corresponds to the method of geometric parallelization. Its implementation is simple, but its disadvantage is unpredictable distribution of work load implying situations when some processors become idle.

Dynamic load balancing aims to share the work load uniformly. However, to implement such a strategy the problems of controlling the load balance and termination detection should be solved.

All earlier mentioned references on parallel branch-and-bound considered combinatorial optimization. The literature on parallel branch-and-bound for global optimization is limited. However parallel branch-and-bound methods for combinatorial optimization and global optimization posses many similarities and few differences, as discussed in [8].

The results of some numerical experiments with an interval method for global optimization on a transputer system are presented in [24]. A parallel method for verified global optimization using a centralized mediator for the dynamic load balancing is proposed in [3].

The use of a depth first strategy in parallel global optimization is discussed in [32].

1.5 Criteria of parallelization

The efficiency of the parallelization is measured using several criteria. A commonly used criterion of parallel algorithms is speedup:

$$s_m = \frac{t_1}{t_m}$$

where t_m is time used by the algorithm implemented on m processors. The speedup divided by number of processors is usually called the efficiency:

$$e_m = \frac{s_m}{m}.$$

Normally, $1 \le s_m \le m$ and $0 \le e_m \le 1$. However in parallel branch and bound the progress of the search may differ for the different number of processors. This depends on:

- initial partitioning,
- how sub-problems are subdivided,
- when better values of the objective function $\overline{f}$ are found and exchanged (sometimes earlier and sometimes later than by the corresponding sequential algorithm).

If the progress of search is different, then the search tree is different, and the total number of expanded subdomains is different. There may be either more or fewer or the same number of expanded subdomains. This may cause anomalies:

- detrimental anomaly s_m<1, more expanded subdomains,
- deceleration anomaly $s_{m_1} > s_{m_2}$, for m_1<m_2, more expanded subdomains when more processors,
- acceleration anomaly s_m>m, fewer expanded subdomains.

The anomalies are mentioned in number of papers about the parallel branch and bound, for example [65], [22]. The performance of the parallel branch-and-bound algorithms with the best first strategy by examining various anomalies was analyzed in [51].

The initial partitioning must not depend on the number of processors to decrease anomalies. However, the large number of initial sub-problems must be generated when various numbers of processors are used. Furthermore, the load balance depends on the number of processors if the initial partitioning does not. This happens since number of initial sub-problems for each processor decreases when the number of processors increases. When the initial partitioning depends on the number of processors and each processor gets the same number of initial sub-problems, the load balance does not depend on the number of processors. In this case the evaluation of criteria is more correct, but anomalies are more likely.

The speedup and the efficiency are not well appropriate criteria in case of anomalies. The criteria of pseudo efficiency and proportion of time spent on calculations (other time is spent on communications or being idle) are less sensitive to anomalies. The criterion of pseudo efficiency is proposed in [65]

$$pe_m = \frac{t_1 / |T_1|}{m \times t_m / |T_m|},$$

where $|T_m|$ is the measure of amount of work done by the algorithm using m processors. The good measure of amount of work must be chosen when the pseudo efficiency criterion is used. In [65] the amount of work is the number of problem-states expanded during the solution. The problem-state is similar to the sub-problem in the global optimization branch and bound.

The total number of calls of the objective function is good measure of the amount of work:

- when the objective function is “expensive”, i.e. its calculation requires more time than the auxiliary calculations by the optimization algorithm,
- when the amount of the auxiliary calculations is proportional to the number of calls of the objective function.

In the other cases the number of calls of the objective function is not a good measure of the amount of work.

The proportion of time spent on calculation is also called processor utilization [9]. It is the average percentage of the total solution time spent by each processor doing useful work as opposed to being idle or waiting for communication or memory access. The processor utilization is always between 0 and 1. It gives some indication on whether the given implementation suffers from problems with workload distribution and communication bottlenecks.

2 Lipschitz global optimization with simplicial partitioning

In this chapter a new Lipschitz global optimization algorithm with simplicial partitioning is presented. The numerical results of experiments with the sequential algorithm are discussed and compared with the results of other Lipschitz global optimization algorithms. The parallelization of the algorithm is described. The parallel implementations are assessed with respect to criteria of parallelization.

2.1 Introduction

The global optimization problem is considered: find $x_{\min} \in X \subseteq R^n$ such that $f_{\min} = f(x_{\min}) \leq \min\limits_{x \in X} f(x) + \varepsilon$, where ε is a small positive constant. The objective function *f* defined over $X \subseteq R^n$ is supposed to be Lipschitzian, i.e.

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

where *L* is a constant called Lipschitz constant, $\left\| \cdot \right\|$ denotes the Euclidean norm.

Lipschitz optimization is one of the most deeply investigated subjects of global optimization. The advantages and disadvantages of this approach are discussed, e.g. in [31], [61]. In this chapter the possibilities of enhancing of efficiency of Lipschitz global optimization branch and bound algorithms is considered introducing new irregular simplex-based branching and bounding techniques.

In Section 2.2 a partitioning method of irregular simplices is proposed. In Section 2.3 the calculations of bounds for Lipshitzian functions are discussed. In Section 2.4 the test results of the proposed simplex-based branch and bound algorithm are given. The proposed algorithm is compared with other branch and bound algorithms according to the criterion of the number of function evaluations. In Section 2.5 implementation and evaluation of parallel algorithms is discussed.

2.2 Simplex-based branching

Simplex is the convex hull of *n*+1 points in *n*-dimensional Euclidean space (*n* is a positive integer). In one-dimensional space a simplex is a segment of line, in two-dimensional space it is a triangle, in three-dimensional space it is a tetrahedron. A simplex is a polyhedron in *n*-dimensional space, which has the minimal number of vertices.

Usually, a feasible region in Lipschitz optimization is defined as a hyper-rectangle. There are two main covering strategies of a hyper-rectangle by simplices: over-covering and face to face vertex triangulation.

Using the first strategy a hyper-rectangle is covered by one simplex. One version is to fit a hyper-rectangle into a simplex matching a vertex. One vertex of the hyper-rectangle and one vertex of the simplex are matched, edges of the simplex from this vertex include edges of the hyper-rectangle from this vertex, and the opposite vertex of the hyper-rectangle is placed on the opposite face of the simplex. The one- and two-dimensional examples are shown in Figure 3. Covering is not unique. When implementing, it is aimed to minimize the over-covering.

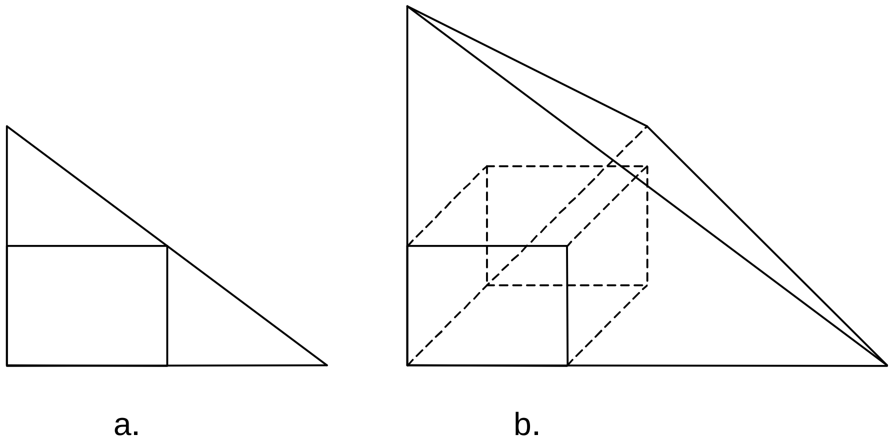


Figure 3. Covering by matching a vertex.

The other version is to fit a hyper-rectangle into a simplex placing a face of the hyper-rectangle on a face of the simplex. The examples of such covering are shown in Figure 4. In this case covering using a regular simplex is possible.

An *n*-dimensional hyper-rectangle is face-to-face vertex triangulated if it is partitioned into finitely many *n*-simplices, where the vertices of *n*-simplices are also the vertices of the hyper-rectangle. The examples of such partition are shown in Figure 5.

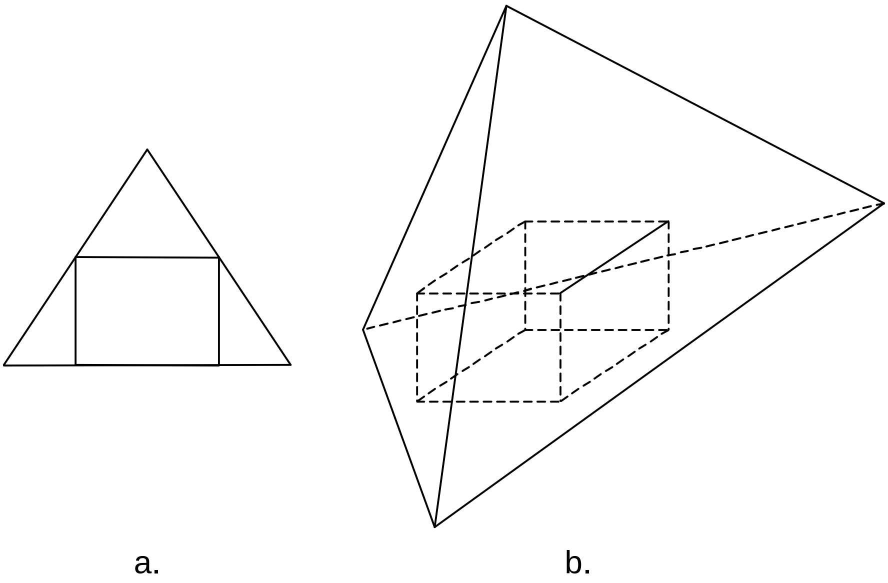


Figure 4. Covering by placing a face of the hyper-rectangle on a face of the simplex.

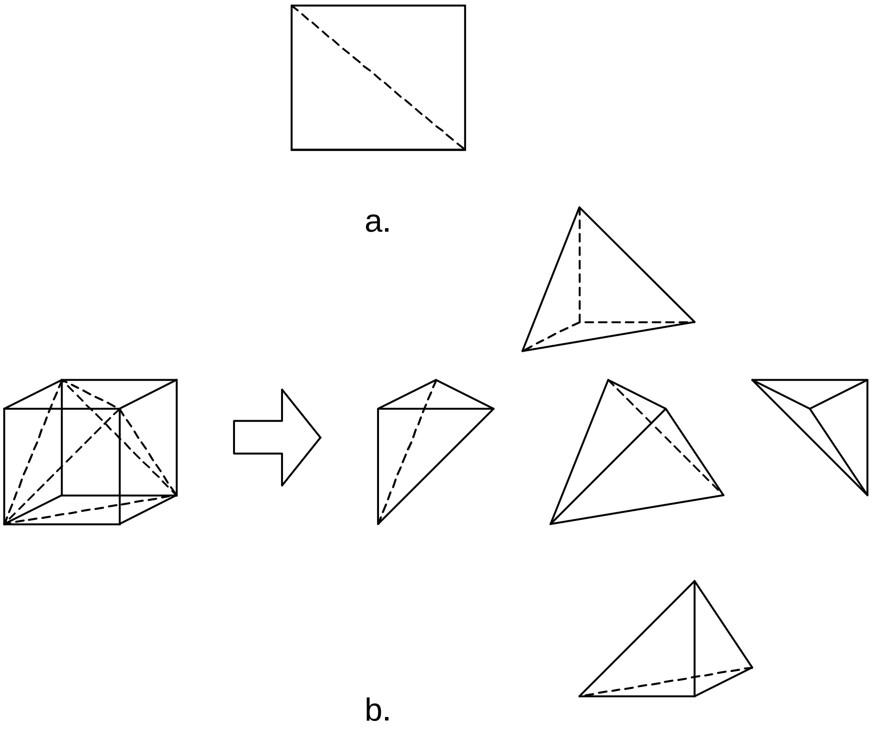


Figure 5. Face to face vertex triangulation of a hyper-rectangle.

Branching is carried out by means of partitioning a simplex into sub-simplices. It is known that tight bounds for function values can not be constructed for a perverted simplex. An irregular triangle (two-dimensional simplex) may be divided into 4 similar triangles, a right equilateral triangle may be divided into 2 similar triangles, see Figure 6. If other branching strategies are used, the perversion of simplices must be prevented. One way of prevention is to divide simplices by a hyper-plane passing through the middle point of the longest edge and the vertices not belonging to the longest edge. This ensures that the longest edge of sub-simplices is not more than two times longer than other edges. The examples of such division are shown in Figure 7.

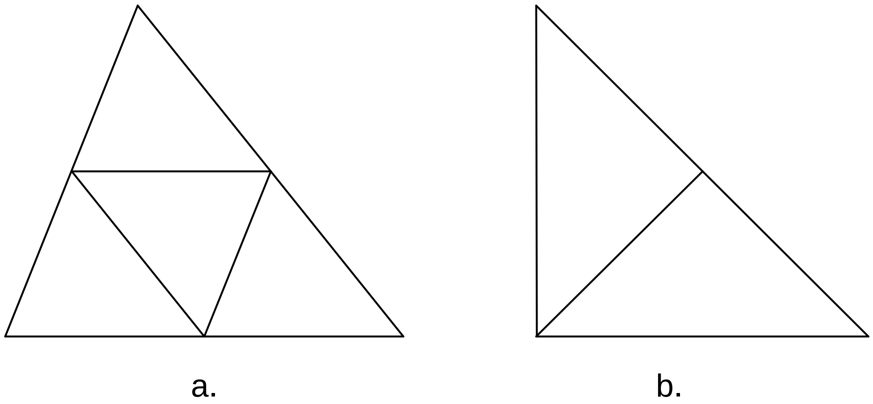


Figure 6. Partition of a triangle into similar triangles.

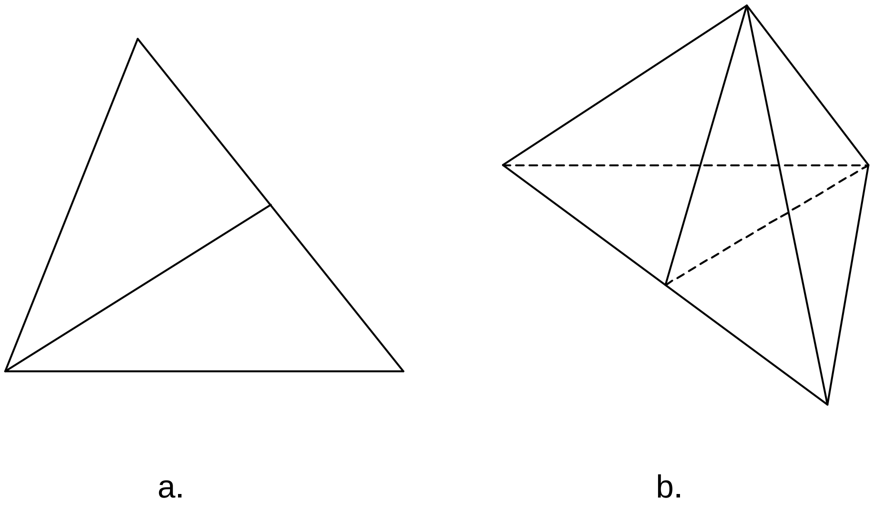


Figure 7. Partition of a simplex into two simplices using the midpoint of the longest edge.

2.3 Bounds for Lipschitz function values on a polyhedron

The efficiency of the branch and bound technique depends on the bound calculation. The algorithm stops faster in case of tight bounds of the minimum value of *f* over polyhedron *I*. However, calculation of tight bounds usually is time consuming. The general algorithm for calculation of upper and lower bounds of the minimum value of *f* over polyhedron *I* is presented in Algorithm 2. First, the points in a polyhedron are chosen and the function values at these points are calculated. The upper bound is the minimum of these values. The lower bound is calculated using these values and the Lipschitz condition.

Algorithm 2. General algorithm for calculating bounds of the minimum value of a function on a polyhedron.

```
Choose a finite set $D$ of points in $I$.
$UB(I) = \min_{x \in D} f(x)$.
Build a lower bounding function $F$ on $I$.
Compute $LB(I) = \min_{x \in I} F(x)$.
```

The lower bounding function is the upper envelope of a set of circular cones with parallel symmetry axes. If $n > 1, \left(I \subseteq R^n\right)$ and more than one point in *D* is used, the minimization of lower bounding function is time consuming. The finite set of local optimum points of such function could be computed by solving systems of quadratic equations [31].

Usually, one point in the interior of a polyhedron or some/all of extreme points (vertices of a polyhedron) are chosen for evaluation of the function. If one point is used, the lower bound is simple to calculate:

$$LB(I) = f(x_o) - L \max_{x \in I} \|x - x_o\|.$$

The lower bound would be more tight if

$$x_o = \arg\min_{x_t \in I} \left(\max_{x \in I} \|x - x_t\| \right).$$

If the polyhedron is simplex and the center point of covering hyper-sphere is in the interior of the simplex, x_o is the center point of covering hyper-sphere. This point is an intersection of hyper-planes passing through the midpoints of and perpendicular to the edges. It can be found by solving a system of linear equations.

The function values at vertices of a polyhedron can be used also when calculating bounds for neighboring and descendant polyhedrons. However, if n>1 and more than one vertex is used, the minimization of lower bounding function is time consuming. The calculations are simpler when one vertex is used. The lower bound is calculated using the function value at this vertex and the length of the longest edge from this vertex. The lower bound is underestimated. The tighter lower bound is the highest lower bound calculated using each vertex:

$$LB(I) = \max_{x_v}\left(f(x_v) - L \max_{x \in I} \| x - x_v \| \right).$$

The algorithm for calculation of such bounds is shown as Algorithm 3.

Algorithm 3. Algorithm for simple calculation of bounds of the minimum value of a function on a polyhedron.

```
$LB$=∞
For all vertices $i$
    $maxl$=0
    For all vertices $j$
        $maxl$=max( $maxl$, $||x_i – x_j||$ )
    $LB$=max( $LB$, $f(x_i)$ – L × $maxl$ )
```

2.4 Numerical results

Thorough theoretical investigation of the proposed version of branch and bound algorithm is difficult, although their convergence is obvious. Therefore, a numerical modeling method was used for the investigation.

The C++ program for simplex-based branch and bound was implemented with three strategies of selection. The latest implementation is available for noncommercial use at [88]. The best first strategy was implemented as priority queue, the depth first strategy as FILO, the breadth first strategy as FIFO.

Three ways of initial covering were implemented: over-coverings as illustrated in Figure 3 and Figure 4 and face to face vertex triangulation as illustrated in Figure 5. Two ways of branching were implemented: through the midpoint of the longest edge as illustrated in Figure 6b and Figure 7 and through the midpoints of all edges as illustrated in Figure 6a.

The bounds of the minimum of the objective function are calculated in one of the two ways described in the Section 2.3: using function value at center point of covering hyper-sphere x_o or function values at one or more vertices x_v.

The number of calls of the objective function is used as a criterion of efficiency. A set of two-dimensional test functions from [31] was used for testing and evaluation.

The test results have shown superiority of calculation of bounds using function values at vertices over using function value at center point of covering hyper-sphere. The disadvantage of the later is non-reusability of the function value at this point for calculation bounds for other simplices. Contrary the function values at vertices are used when calculating bounds for neighboring and descendant simplices.

The experiments have confirmed that it is unlikely worth to evaluate function at all vertices of a simplex. The best efficiency has been reached when bounds are calculated as follows. If function values at vertices have not been evaluated before, then a function value is evaluated at vertex not belonging to the longest edge. It is used to calculate bounds. In other case all earlier evaluated function values at vertices of the simplex are used to calculate lower bound.

The best first strategy is more efficient than the others, although efficiency of the breadth first strategy is similar. The efficiency is worst for the depth first strategy. The experiments have shown that it is better to branch simplices through the middle point of the longest edge, but not through the midpoints of all edges. The way of covering does not have much influence, although superiority of vertex triangulation is more frequent than that of over-coverings.

Typical results of minimization one of the test functions from [31] using different versions of the proposed algorithm are given in Table 1. The bounds are calculated using function values at vertices. Similar results are obtained for other test functions.

Table 1. Results of minimization one of the test functions from [31] using different versions of the proposed algorithm.

Initial covering	Branching	Selection	Number of function calls
A face to face vertex triangulation	Two descendents as in Figure 6b and Figure 7	Best first	244
		Breadth first	244
		Depth first	259
	Four descendents as in Figure 6a	Best first	371
		Breadth first	384
		Depth first	658
Regular over-covering, as in Figure 4	Two descendents as in Figure 6b and Figure 7	Best first	263
		Breadth first	262
		Depth first	427
	Four descendents as in Figure 6a	Best first	328
		Breadth first	328
		Depth first	321
Over-covering, as in Figure 3	Two descendents as in Figure 6b and Figure 7	Best first	257
		Breadth first	260
		Depth first	275
	Four descendents as in Figure 6a	Best first	363
		Breadth first	383
		Depth first	401

The testing results suggest the following version of the *n*-dimensional simplex-based branch and bound algorithm to be used:

- To cover a feasible region by simplices using face to face vertex triangulation.
- To partition the simplices by a hyper-plane passing through the midpoint of the longest edge and the vertices whose do not belong to the longest edge.
- To estimate the bounds of the minimum of the function over the simplex using function values at one or more vertices.
- To use the best first strategy of selection.

The final sequential algorithm is shown as Algorithm 4.

Algorithm 4. Sequential algorithm.

```
CoverRegion( FeasibleRegion );
While( WorkPool.IsNotEmpty() )
     Branch( WorkPool.Best() );
```

The best branch and bound algorithm from [31] denoted as GHJ and the two-dimensional branch and bound algorithm with regular simplex-based covering from [12] denoted as CZ were used for comparison. The results of optimization the test functions using GHJ, CZ and proposed algorithms are presented in Table 2.

Table 2. Numbers of function calls minimizing test functions using different algorithms.

Test function from [31]	GHJ [31]	CZ [12]	proposed
1	643	489	611
2	167	37	132
3	3531	2618	2185
3.1	3953	3245	1559
3.2	3035	2665	1225
3.3	3689	3387	1429
4	45	41	70
5	73	53	80
6	969	629	838
7	7969	6370	3117
8	301	255	244
9	13953	8759	3773
9.1	14559	9531	3890
9.2	13281	9002	3729
9.3	12295	8917	3610
10	1123	820	848
11	2677	2222	1566
12	12643	10851	4001
13	15695	10643	4084

The numbers of function calls of minimization most of test functions is smallest for the proposed algorithm. The numbers of function calls of other test functions are similar. The results of experimental testing show that the proposed algorithm performs better than other branch and bound algorithms for Lipschitz optimization.

2.5 Parallel algorithms

The rules of covering, branching, bounding and selection by parallel algorithms are the same as by the sequential algorithm:

- A feasible region is covered by simplices using face to face vertex triangulation.
- The simplices are branched by a hyper-plane passing through the midpoint of the longest edge and the vertices whose do not belong to the longest edge.

- The lower and upper bounds of the minimum of the function over the simplex are estimated using function values at vertices.
- The best first selection strategy is used.

Three parallel branch-and-bound algorithms are considered, each of them uses one of the load balancing strategy:

- Algorithm with static load balancing,
- Master-slave algorithm with centralized load balancing,
- Algorithm with distributed dynamic load balancing.

2.5.1 Algorithm with static load balancing

By the static load balancing, the tasks are initially distributed and then the processors work independently and do not exchange any later generated tasks. Each parallel processor runs the same algorithm, which is shown as Algorithm 5. The algorithm is very similar to the Sequential Algorithm 4. The differences are:

- Each processor covers one part of the feasible region. This is shown symbolically using division by the number of processors, *m*.
- After completion the results of optimization are collected.
- Processors interchange the best currently found values of the objective function when they are found.

Algorithm 5. Parallel algorithm with static load balancing.

```
CoverRegion( FeasibleRegion / m );
While( WorkPool.IsNotEmpty() )
     Branch( WorkPool.Best() );
CollectResults();
```

The algorithm is easy to implement. However the load balance may be poor since the processors may finish their work at different times.

2.5.2 Master-slave algorithm

In the parallel master-slave system there is one master processor, which controls the optimization process. The slave processors receive tasks from the master, perform computations on them and send the results to the master. Slaves perform computations on promising subproblems. The master controls load balancing and termination.

The algorithm is shown as Algorithm 6. The master processor holds the work pool – the collection of subproblems. When the slave processor has no work it requests a task from the master processor. The master processor sends out the best subproblem it has. When the slave finishes the task it sends the results to the master processor.

Algorithm 6. Parallel master slave algorithm.

```
If( Rank == Master )
     CoverRegion( FeasibleRegion );
     While( WorkPool.IsNotEmpty() or NotAllRequested() )
          Receive( Request, fromAny );
          If( WorkPool.IsNotEmpty() and SomeRequested() )
               Send( WorkPool.Best(), toRequested );
          While( Receive( Task, fromAny )
               WorkPool.Add( Task );
     Send( Stop, toAll );
Else
     Send( Request, toMaster );
     While( not Receive( Stop, fromMaster ) )
          If( Receive( Task, fromMaster ) )
               Branch( Task );
               While( WorkPool.IsNotEmpty() )
                    Send( WorkPool.Best(), toMaster );
               Send( Request, toMaster );
CollectResults();
```

The master processor manages the termination. The termination conditions are:

- The work pool is empty.
- Every slave processor requested for a task. This means that all slaves have finished their tasks and have sent the results to master. If any slave processor is running it is possible that it will generate new tasks.

The slaves terminate when they receive the termination message from the master. After completion the results of optimization are collected. All processors interchange the better values of the objective function when they are found.

The disadvantage of this scheme is that master can become a bottleneck especially when the system is large and the communications are frequent and heavy. The slaves can become idle when waiting for new tasks.

2.5.3 Algorithm with distributed dynamic load balancing

The distributed parallelization means a cooperative work of a set of equal communicating processors. Dynamic load balancing aims to share the work load uniformly. The main problems of the distributed dynamic load balancing are the load balancing and the termination detection.

The algorithm is shown as Algorithm 7. The distributed termination method is based on request and acknowledgement messages. Each processor is in one of the states – active or inactive. Initially a processor has no tasks and is inactive. It becomes active when it gets a task from another processor. The activating processor becomes the parent. In this way a tree of active processors is created. Each active processor has the unique parent, which is defined by the variable `Parent` in the Algorithm 7. For inactive processors `Parent = Rank.`

The active processor may receive tasks from other processors. The received tasks from non-parent processors are acknowledged by the acknowledgement message. The processor sends acknowledgement message to its parent when it becomes inactive. It becomes inactive when:

- It has finished all its tasks. Local termination condition.
- It has sent all its acknowledgement messages for tasks it has received.
- It has received all the acknowledgment messages for tasks it has sent.

The process becomes inactive before its parent process. When the root process becomes inactive the algorithm terminates.

Two initialization strategies are used: distributed and centralized. The distributed strategy is similar to the case with static load balancing. Each processor covers one part of the feasible region. This is shown symbolically using division by the number of processors *m*. The root processor after initialization becomes a parent of all processors.

The centralized initialization strategy is similar to the case with centralized load balancing. The root processor covers the feasible region. Other processors are inactive at the beginning and have no parents.

Algorithm 7. Parallel algorithm with distributed dynamic load balancing.

```
Sons[0..m-1] = 0;
If( Init0 )
     CoverRegion( FeasibleRegion / m );
     Parent = Root;
     If( Rank == Root )
          Sons[1..m-1] = 1;
Else
     If( Rank == Root )
          CoverRegion( FeasibleRegion );
     Parent = Rank;
While( not Received( Stop, fromRoot ) )
     If( WorkPool.IsNotEmpty() )
          MyTask = WorkPool.Best();
     Else
          MyTask = NULL;
          Send( Request, toRoundRobin() );
     While( Receive( Anything, fromAny ) )
          If( Received( Request, fromAny ) )
               If( WorkPool.IsNotEmpty() )
                    Send( WorkPool.Best(), toSource );
                    Sons[ Source ]++;
               Else
                    Send( Rejection, toSource );
          If( Received( Acknowledgement, fromAny )
               Sons[ Source ]--;
          If( Received( Task, fromAny ) )
               If( NoParent() )
                    Parent == Source;
               Else
                    Send( Acknowledgement, toSource );
               If( MyTask == NULL )
                    MyTask = Task;
               Else
                    WorkPool.Add( Task );
     If( MyTask != NULL )
          Branch( MyTask );
     If( WorkPool.IsEmpty() and NoSons() )
          If( Rank == Root )
               Send( Stop, toAll );
          Else If( Parent() )
               Send( Acknowledgement, toParent );
               Parent = Rank;
CollectResults();
```

Receiver initiated, or “on demand”, load balancing strategy is used. When the processor has no tasks it requests a task from the other processor. The general structure is used – all processors are equal candidates and any processor could be selected for communications. The round robin algorithm is used to select the candidate processor in sequential order coming back to the first processor after the last one.

Using the distributed dynamic load balancing the work could be distributed uniformly. However communications between processors cause an overhead during execution, because some time is spent on communications. Sometimes there could not be enough tasks to share.

The Algorithm 7 with distributed dynamic load balancing is much more complicated than Algorithm 5 with static load balancing and master-slave Algorithm 6. The implementation of this algorithm is much more difficult and sometimes could be not worth the time spent on implementation.

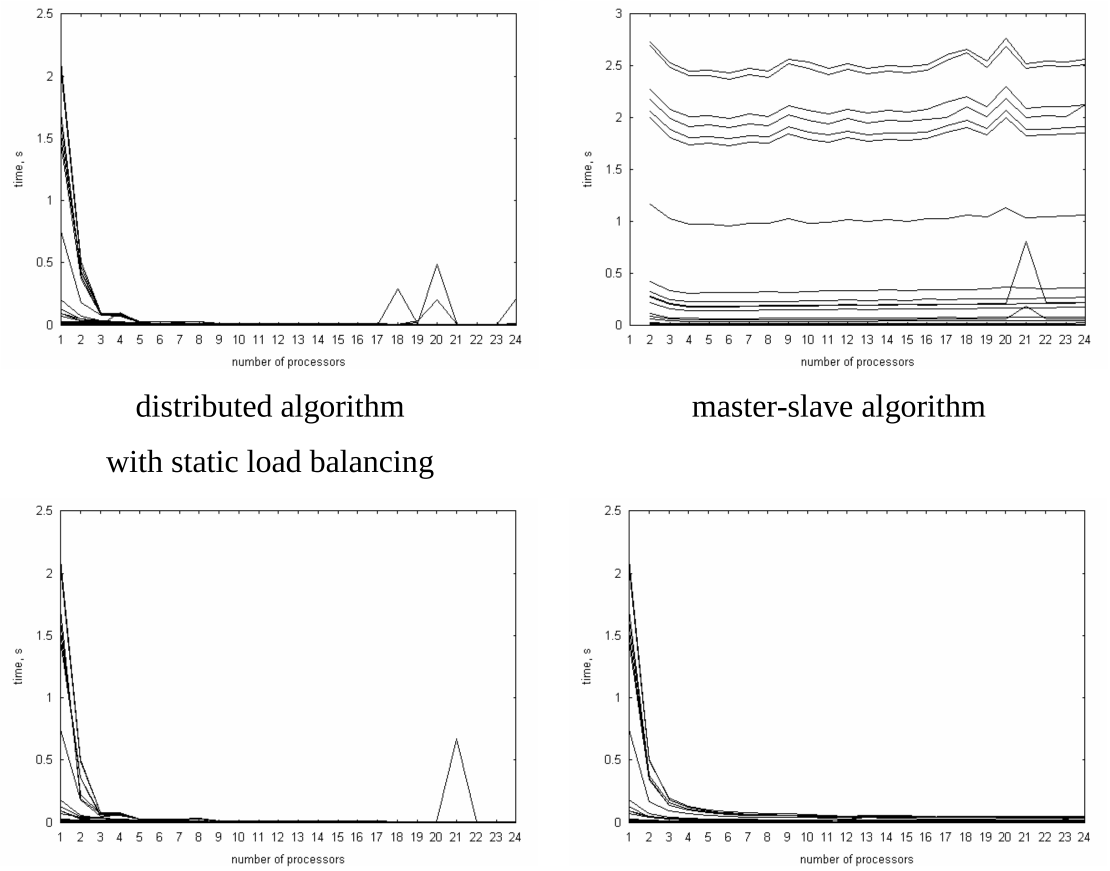
2.5.4 Results of experiments with parallel algorithms

All three parallel algorithms have been implemented in C++ and MPI (Message-Passing Interface – a standard specification for message-passing libraries). The latest implementation is available for noncommercial use at [88]. The algorithms have been tested on the Sun HPC 6500 UltraSPARC-II based system with Sun MPI at Edinburgh Parallel Computing Center. Up till 24 parallel processors have been used.

The same test functions were used in testing and evaluation of criteria as in the sequential case. The total times of optimization of every test function for the different number of processors are shown in Figure 8. In this and following figures there are four diagrams:

- The left upper diagram represents results of algorithm with the static load balancing.
- The right upper diagram represents results of master-slave algorithm. There must be not less than two processors: one master and one slave.
- The left lower diagram represents results of the algorithm with the distributed dynamic load balancing and the distributed initialization.
- The right lower diagram represents results of the algorithm with the distributed dynamic load balancing and the centralized initialization.

Every curve in the diagrams represents the results of optimization of the different test function from [31].



distributed algorithm with static load balancing

master-slave algorithm

distributed algorithm with dynamic load balancing and distributed initialization

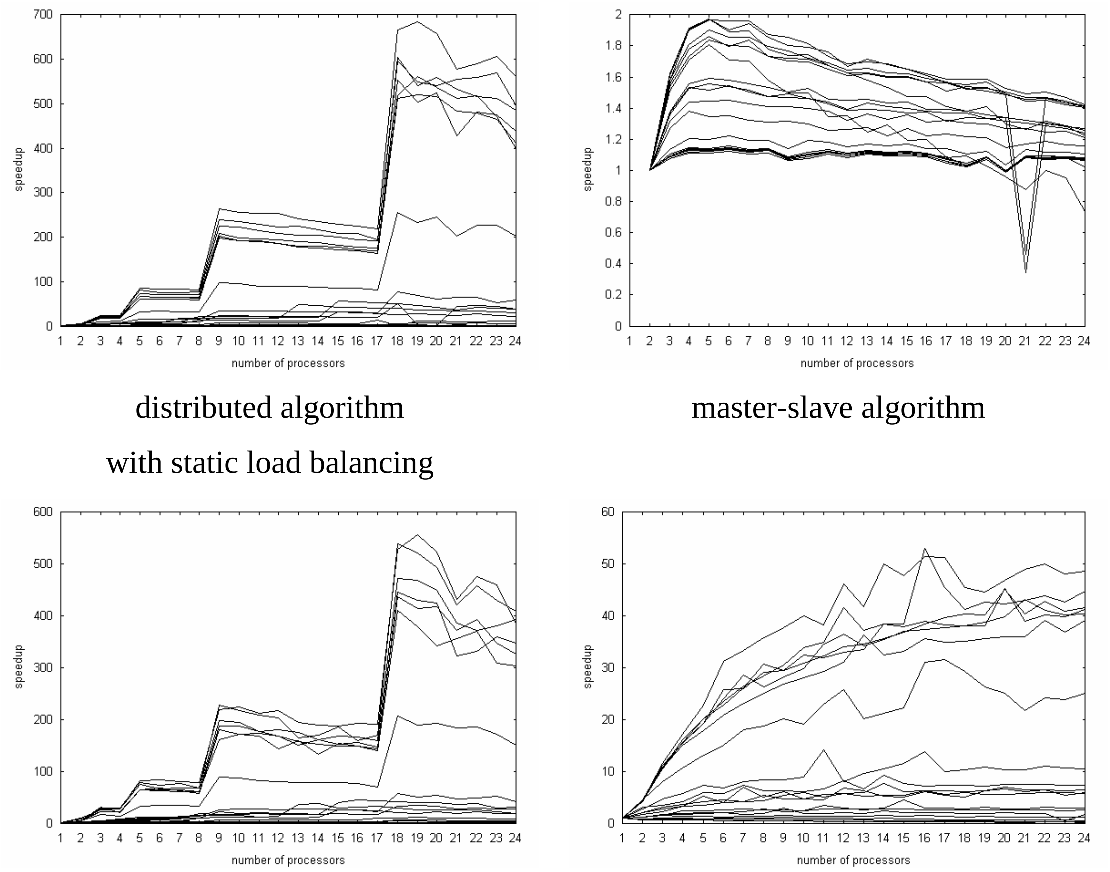
distributed algorithm with dynamic load balancing and centralized initialization

Figure 8. Times of optimization of the test functions.

The times of optimization show that the master-slaver paradigm is not proper in our case. The times of optimization decrease slowly and only up to 4 processors, then the times increase slightly. The time of computations on one task is not big comparing with the time of communications needed to send the task and to receive the results. The communications are frequent. The master becomes a bottleneck, the slaves become idle when waiting for new tasks. Another reason may be the shortage of tasks.

The performance of all distributed algorithms is better. The times of optimization decrease fast. The time diagrams do not show the differences between distributed algorithms.

The diagrams of criteria of parallelization: speedup s_m and efficiency e_m diagrams for the different number of processors are shown in Figure 9 and in Figure 10.



distributed algorithm
with static load balancing

master-slave algorithm

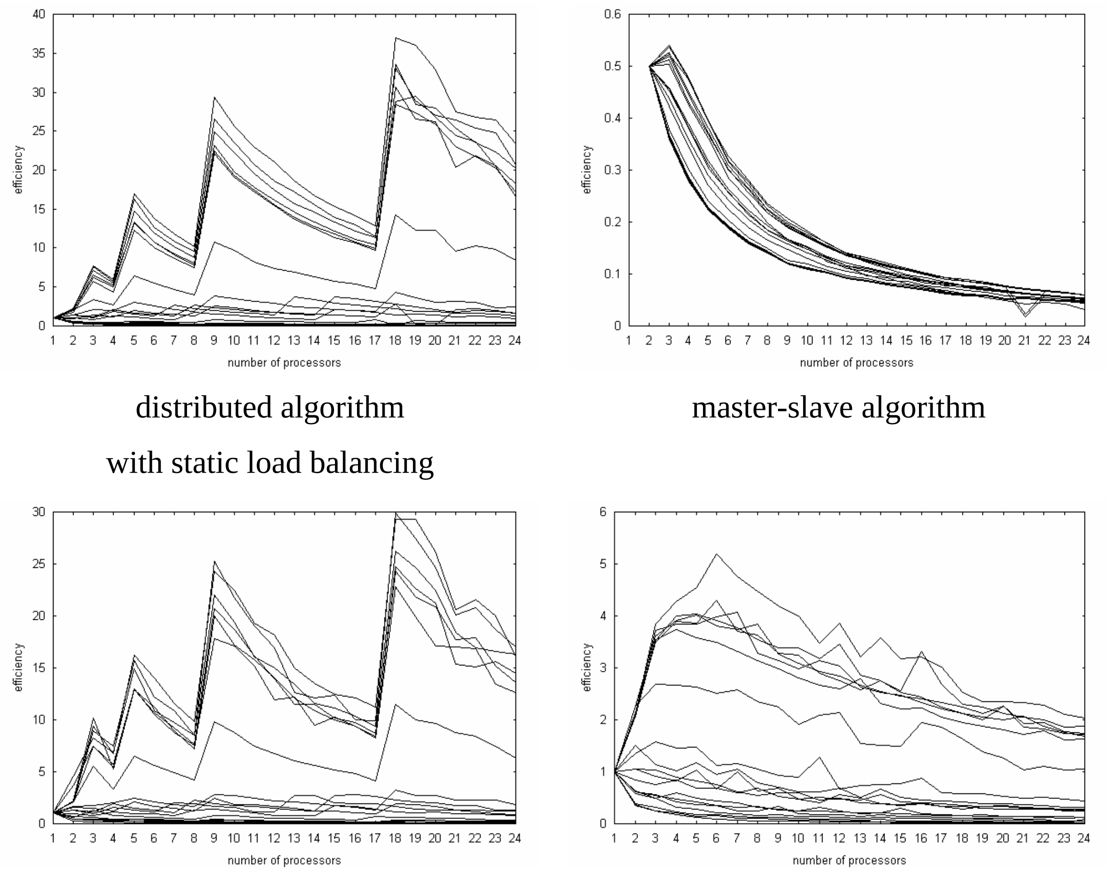
distributed algorithm with dynamic load balancing and distributed initialization

distributed algorithm with dynamic load balancing and centralized initialization

Figure 9. Speedups of parallel algorithms.

The diagrams show that the deceleration anomaly $s_{m_1} > s_{m_2}$, for $m_1<m_2$, exists for the master-slave algorithm. This is because the master-slave paradigm is not proper in our case because the master becomes a bottleneck.

The acceleration anomaly $s_m>m$ exists for distributed algorithms. It gives large steps in diagrams in the left. These diagrams represent algorithms with the static load balancing and with the distributed dynamic load balancing and the distributed initialization. The initialization for both these algorithms is similar. Each processor covers one part of the feasible region. The initial partitioning depends on the number of processors and the total number of expanded subdomains is different. This is because the progress of search is different, the better values of the objective function $\overline{f}$ are found at different times.



distributed algorithm with static load balancing

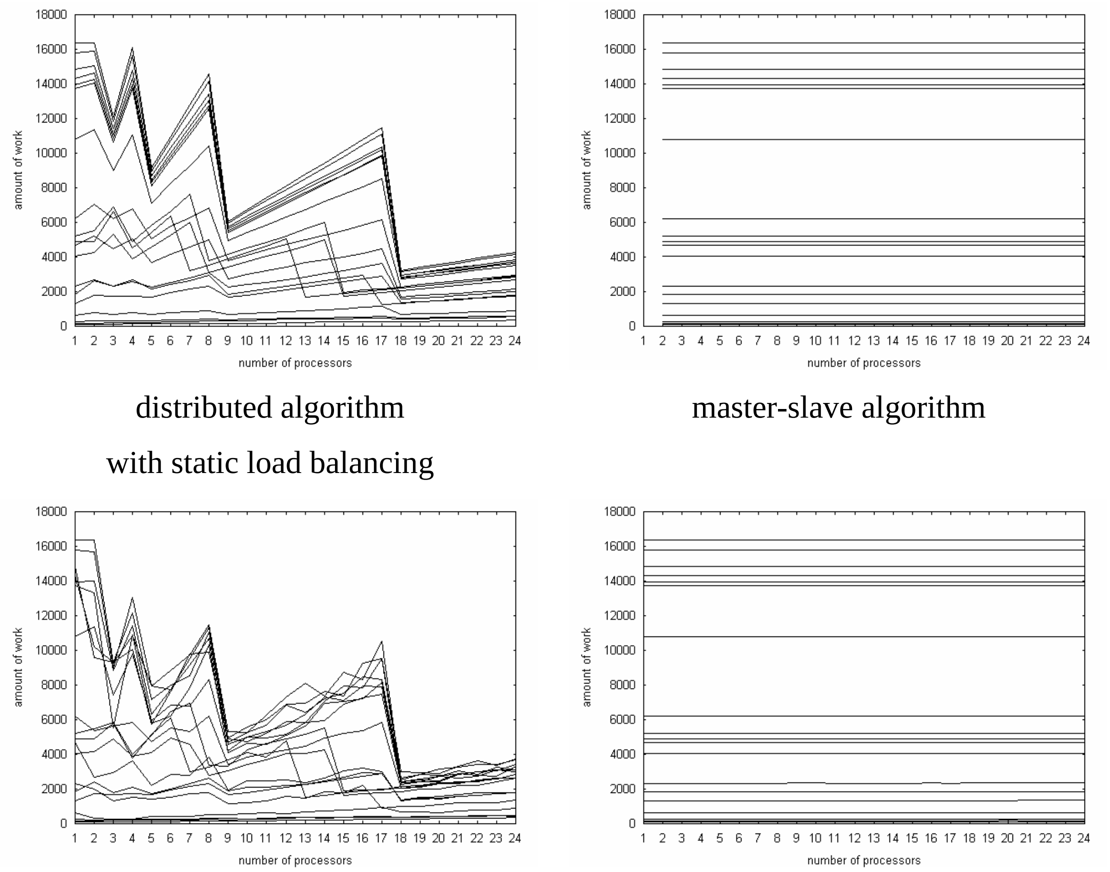
master-slave algorithm

distributed algorithm with dynamic load balancing and distributed initialization

distributed algorithm with dynamic load balancing and centralized initialization

Figure 10. Efficiencies of parallel algorithms.

The speedup and the efficiency are not appropriated criteria when anomalies exist. The pseudo efficiency criterion pe_m is less sensitive to anomalies. The good measure of amount of work must be chosen when the pseudo efficiency criterion is used. As a measure of the amount of work the total number of calls of the objective function is used. The total numbers of calls of the objective function during optimization of the test functions for the different number of processors are shown in Figure 11.



distributed algorithm with static load balancing

master-slave algorithm

distributed algorithm with dynamic load balancing and distributed initialization

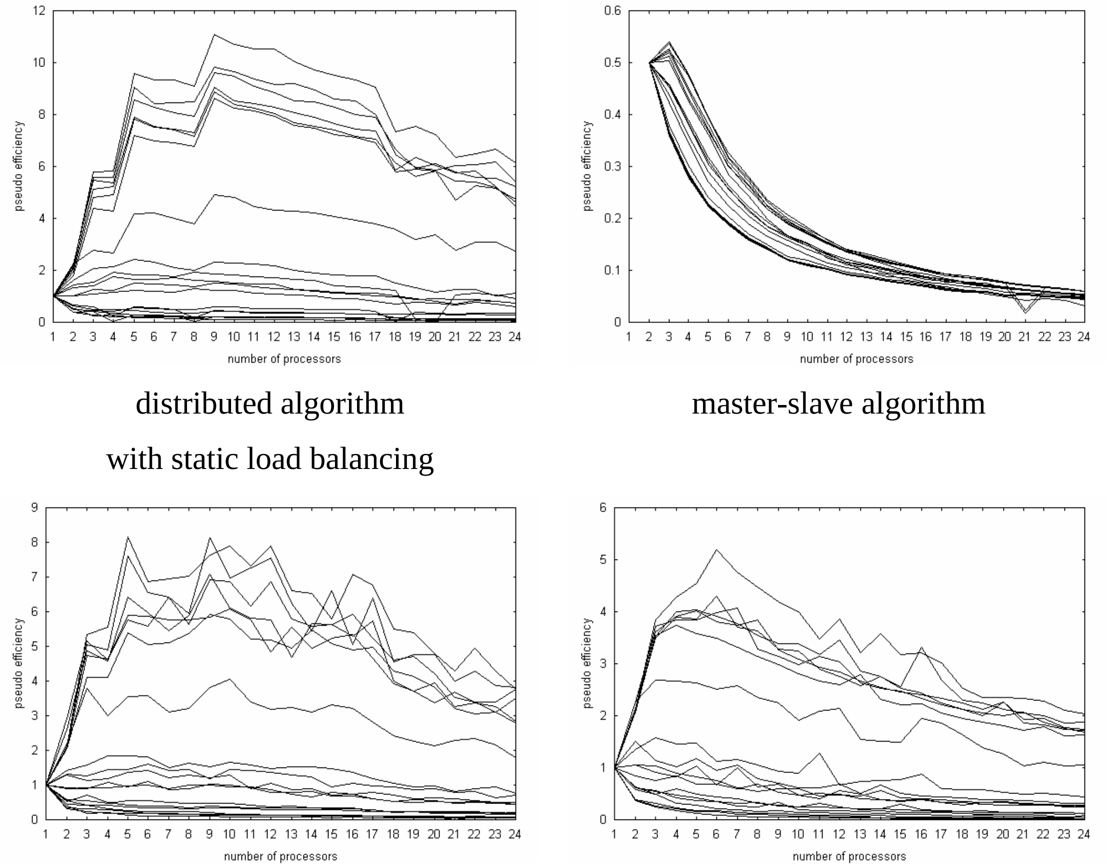
distributed algorithm with dynamic load balancing and centralized initialization

Figure 11. Total numbers of calls of the objective function during optimization.

The diagrams confirm the reason of acceleration anomaly for algorithms with the distributed initialization. The initial partitioning depends on the number of processors and the total number of expanded subdomains is different. The changes in the diagrams in the left are similar to diagrams of efficiencies in Figure 10.

The total numbers of calls of the objective functions during optimization using the master-slave algorithm is constant. For the distributed algorithm with dynamic load balancing and centralized initialization the amount of work changes very slightly.

The pseudo efficiency diagrams for the different number of processors are shown in Figure 12. Although the influence of different amount of work is overcome using the pseudo efficiency criterion, acceleration still exist in the performance of all the distributed algorithms. The possible reason of this may be the wrong choice of the amount of work. Another reason may be the non-homogeneity of the system, for example the different speeds of the processors and communications.



distributed algorithm with static load balancing

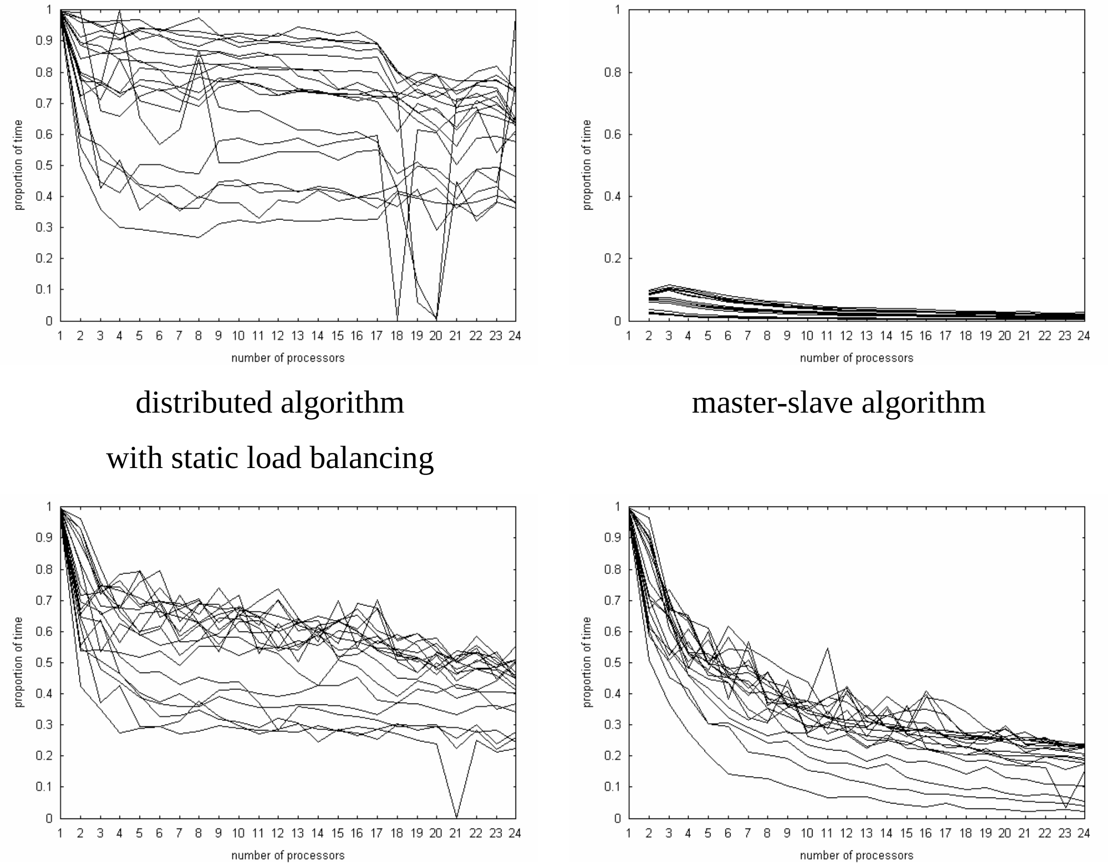
master-slave algorithm

distributed algorithm with dynamic load balancing and distributed initialization

distributed algorithm with dynamic load balancing and centralized initialization

Figure 12. Pseudo efficiencies of parallel algorithms.

Because of anomalies the shown diagrams of criteria of parallel algorithms are not well suited for the comparison of the parallel algorithms. The diagrams do not show clear superiority of any of the distributed algorithms. More information could be extracted from the proportion of time spent on calculation or processor utilization. It gives some indication on whether the given implementation suffers from problems with workload distribution and communication bottlenecks. The diagrams of the proportions of time spent on calculation are shown in Figure 13.



distributed algorithm
with static load balancing

master-slave algorithm

distributed algorithm with dynamic load balancing and distributed initialization

distributed algorithm with dynamic load balancing and centralized initialization

Figure 13. Proportions of time spent on calculation.

The diagrams show that the processor utilization of the master-slave algorithm is very bad. Less than one tenth of time is spent on calculations. Other time is spent on communications or being idle.

When 15 processors are used the average processor utilization of the algorithm with static load balancing is 0.6. This means that when 15 processors are used the algorithm makes 15×0.6=9 times more calculations than when 1 processor is used. When 15 processors are used the distributed algorithm with dynamic load balancing and distributed initialization makes on average 15×0.5=7.5 times more calculations than when 1 processor is used. When 15 processors are used the distributed algorithm with dynamic load balancing and centralized initialization makes on average 15×0.3=4.5 times more calculations than when 1 processor is used.

Except of some changes for the worse of processor utilization when the number of processors is 18-21, the algorithms with distributed initialization are utilized better. This is possibly because of the shortage of tasks when the centralized initialization is used. Only the root processor has tasks after the initialization while others are idle waiting for them. Besides the search tree is narrower than in the case of distributed initialization. However the search progress almost does not depend on the number of processors when the centralized initialization is used and the evaluation of criteria of parallel algorithms is more correct.

Although the worst case of processor utilization of the algorithms with static load balancing and with dynamic distributed load balancing with distributed initialization are similar, the average processor utilization of the algorithm with static load balancing is better. This shows that there are not enough tasks to share and the load balance is not good. The communications cause an additional overhead and the load balance is improved away.

2.6 Conclusions

The general simplex-based branch and bound algorithm for Lipschitz optimization has been proposed. The rules of selection, covering, branching and bounding have been justified by results of experimental investigations. The efficiency of the algorithm, using the number of function evaluations required to solve the problem, has been evaluated and compared with other branch and bound algorithms for Lipschitz optimization.

The parallel master-slave and distributed algorithms with static and dynamic load balancing have been implemented using C++ and MPI. The criteria of parallel algorithms have been evaluated and discussed. The superiority of distributed algorithms has been shown.

3 Black box global optimization inspired by interval methods

In this chapter a new black box global optimization method inspired by interval arithmetic is presented. Sequential algorithm is implemented and investigated. The results of experimental testing are compared with the results of interval global optimization algorithm [43]. Parallel version of the algorithm is implemented.

3.1 Introduction

Theoretically a solution to a global minimization problem may be found with a prescribed accuracy in case of bounded rate of change of the function values, e.g. Lipschitz continuous functions, with a priori known Lipschitz constant. However, such methods lose their theoretical advantages in a black box situation, the dimensionality of their successful practical applications is rather modest [31]. Interval analytic methods have guaranteed convergence to the set of global minimizers, and they are efficient for a large class of problems (see e.g. [30], [33]). However, the interval methods can not always be applied automatically, e.g. for a problem given by a code not aimed specially for interval arithmetic. This is discussed for instance in [46] where a non-deterministic version of the branch and bound strategy used by interval methods is presented.

The proposed black box optimization method is inspired by the interval methods but the interval arithmetic based procedures are substituted by the heuristic procedures. The method can be interpreted as a strategy for managing local searches in search for global minimizers. Once a local minimizer z has been found, a domain around z is left out from further search, thus we avoid several descents to the same local minimizer. We do not intend to guarantee that the solution will be found with prescribed accuracy, thus the user should specify limits of storage and computer time rather than some accuracy tolerance as required by many traditional approaches.

3.2 A Method

In this section, an algorithm inspired by the interval branch and bound algorithm is presented, for the earlier versions see [45], [46].

The algorithm uses values of the objective function *f* calculated at a number of points in the box *B*. Information on *f'* normally is obtained via finite differences. It is supposed that a sufficiently efficient algorithm to find local minimizers of *f* is available. Our method intends to coordinate local searches in a proper way. When a local minimizer *z* has been found in *B*, a ball $ball_z = \{x \in B \| \|x - z\| \leq \varepsilon_{cluster}\}$ is left out from further consideration, where $\varepsilon_{cluster}$ is a user provided parameter. Thus the candidate set consists of boxes where balls with known local minimizers have been left out. For ease of notation we still refer to such a sub-domain as a *box*.

Since the solution may be lost the algorithm is embedded into a loop which restarts the calculation. We keep an *outer* candidate set *G*, in addition to *C*. When the model algorithm has finished, then the elements of *G* cover *D* except of the balls around the known local minimizers. In the outer iteration *G* forms the basis for a re-generation of the inner candidate set *C*. For a box *B* in *G* the available information about the local minimizers of *B* is recorded to control the split of *B* into several boxes later. *B* is added to *C*. The recorded parameters determining the number of points used during the tests of the reduction phase in may be updated in order to increase the reliability of the method.

The structure of the new algorithm is shown as Algorithm 8. The heuristic counterparts of lower bounds and monotonicity test are introduced. The lower bound reduction is embedded into the Newton-reduction of the reduce-or-subdivide procedure. The boolean *inner-stop* is true when *C* contains no boxes (i.e. only points). The outer stopping condition is a time limit set by the user. Other stopping conditions might be adopted in case of additional information on the problem. The variable *SolSet* is intended to converge towards the set of global minimizers, X^*. The positive number μ is intended to distinguish X^* from other local minimizers. In theory μ might be 0, but in practice (i.e. when rounding errors are present) μ has to be positive in order that *SolSet* eventually will contain X^*.

Algorithm 8. Black box global optimization algorithm inspired by interval arithmetic.

```
initialize: $C$, $\overline{f}$, $G$
while not time-limit exceeded do
    while not inner-stop do
        remove-best ($C$)$\rightarrow B$
        generate sample points
        reduce-or-subdivide ($B$) $\rightarrow result$, $\overline{f}$, $garbage$
        $C \cup \{result\} \rightarrow C$
        $G \cup \{garbage\} \rightarrow G$
    end
    $\left\{p \in C \,\middle|\, f(p) \le \overline{f} + \mu\right\} \cup SolSet \rightarrow SolSet$
    while $G \ne \varnothing$ do
        remove-one ($G$)$\rightarrow B$
        subdivide ($B$)$\rightarrow result$
        $C \cup \{result\} \rightarrow C$
    end
    perhaps adjust reduction parameters
end
```

The details of the algorithm are now explained.

- remove-best (*C*): The box *B* from *C* which has the smallest known function value is chosen.
- Sample points. The sample points are used in the monotonicity and Newton tests. Two strategies are applied for choosing these points. First we use the following regular distribution of up to 2*n*+1 points: One point at *m*(*B*), the center of *B*, and two points for each coordinate direction, $m(B) \pm \frac{1}{3} w(B^{(j)}) * e_j$, $j = 1, \ldots, n$ where $B = B^{(1)} \times \cdots \times B^{(n)}$, *w*(*B*) means the width of the box *B*, and e_j is the *j*-th unit vector. The constant $\frac{1}{3}$ is chosen because it provides a good coverage of *B* in the sense of disjoint balls with centers at the regular sample points. Secondly, we may use random points uniformly distributed in *B*. The total number of sample points, $p_1 = n_{sample}$, has to be provided by the user. We use random points if n_{sample} is greater than 2*n*+1.
- The reduce-or-subdivide procedure is summarized as Algorithm 9, which is explained later.

Algorithm 9. Reduce-or-subdivide procedure.

```
**if** monotone **then**
    Monotone $(B) \to result$
    $B \to garbage$
**else if** (*case 1*) or (*case 2*) **then**
        Newton $(B) \to result$
        $B \setminus \{ball_z | z \in B\} \to garbage$
    **else if** (*case 3*) and ($LB(B) > \overline{f}$) **then**
            $\varnothing \to result$
            $B \setminus \{ball_z | z \in B\} \to garbage$
        **else**
            subdivide $(B) \to B^1, B^2$
            $\{B^1, B^2\} \setminus \{ball_z | z \in B\} \to result$
            $\varnothing \to garbage$
update $\overline{f}$
```

- *G* is emptied in the second inner **while**-loop of Algorithm 8. After *B* has been removed from *G* it is subdivided.
- Subdivision. We always split *B* into two by a hyperplane which is orthogonal to one of the coordinate directions. The result consists of the two new parts. The subdivision is done as follows:

  *Case a.*No known local minimizer in *B*: The splitting divides *B* into two equal parts separated by a hyperplane through the center of *B* and perpendicular to the side of the maximal length.

  *Case b.*One known local minimizer in *B*: Find the coordinate for which the distance from this point to a side of *B* is maximal. The splitting plane is chosen to be perpendicular to this coordinate axis, and it halves the distance mentioned.

  *Case c.*More than one known local minimizer in *B*: Find the two local minimizers with smallest function values. Find the coordinate for which the distance between these two points is maximal. The splitting is made with the plane which is perpendicular to this coordinate direction and has the same distance to the two local minimizers.

The details of the reduce-or-subdivide procedure:

- The monotonicity test is based on the gradient information at the sample points (explicitly calculated if possible, by means of finite differences otherwise). If a local minimizer has previously been found in *B*, however, then *f* cannot be strictly monotonous in *B*, and hence the test is skipped. Otherwise we examine the gradients. If there exists at least one coordinate x_i for which $\partial f(x)/\partial x_i$ has constant sign at all sample points then we decide that no local minimizer exists in *B*. $B \cap \partial_i(D) \rightarrow result$ where $\partial_i(D)$ denotes the facet of *D* (box of lower dimensionality), which is orthogonal to x_i axis, and towards which *f* decreases. The variable *garbage* is set to *B*.

- Newton test is performed if the monotonicity test has not reduced *B*. Local searches for finding local minimizers of *f* in *B* are started from each sample point. The local technique used in our tests is a version of Powell's Dog-Leg Method [64]. The gradients are estimated using forward differences. Notice that balls, $ball_z$, around the local minimizer *z*, which are known at the beginning of the current Newton test, are considered "outside of *B*".
- We distinguish the following four cases:

  Case 1. Every iteration sequence has an iterate which is outside of *B*. In this case we decide that no local minimizer exists in *B*, and we let $B \cap \partial(D) \rightarrow result$ where $\partial(D)$ denotes the border of *D*.

  *Case 2.*All iteration sequences provide convergence to the same point $z \in B$. Then $\{z\}\rightarrow$*result*. (In practice a Newton iteration is stopped by one of the usual stopping rules for local searches (to prevent infinite iterations), and two stopping points are considered equal if their distance is less than $\varepsilon_{cluster}$).

  *Case 3.*Several local minimizers have been found in *B*. In this case the lower bound reduction may take place. The lower bound *LB*(*B*) is estimated using the maximum gradient norm *maxgrad* obtained during the local searches in *B*. Let $f(x_{\min})$ be the smallest known function value in *B*, then

$$LB(B) = \min\left\{ f(x_{\min}) - \max grad * \|x - x_{\min}\| \middle| x \in B \right\}.$$

  If $LB(B) > \overline{f}$ then $\varnothing\rightarrow$*result* otherwise subdivision takes place. (We decided only to use the lower bound reduction in Case 3 because our experiments indicate that this is more efficient).

  *Case 4.*None of the first 3 cases has occurred and subdivision takes place.

Notice that if the local search has the following property: "If there is no local minimizer in *B* then eventually an iterate outside of *B* will be generated", then boxes without local minimizers will always lead to Case 1. Thus the number of boxes in *C* with full dimensionality and without known local minimizers must be very limited. Hence the monotonicity test is most often skipped. Furthermore, the distance between known local minimizers is at least $\varepsilon_{cluster}$.

Assuming some regularity conditions, the convergence of the method to the set of global minimizers is proved in [45]. The key observation in the proof is that no box can remain large during the iteration, i.e. all the boxes in the candidate set will be small when the algorithm has been running for long enough time.

3.3 Testing

Although testing is very important for assessment of new optimization algorithms, a well justified testing methodology is not developed for all classes of optimization methods. For example, detailed theory approves the structure and features of local descent algorithms for smooth objective functions. Specific difficulties of application of such algorithms are well known, and collections of test problems representing important features of practical problems as well as specific difficulties are available [21], [67]. Therefore, a new algorithm of smooth local optimization may be approved by **competitive** testing [34] using a standard methodology. A similar development level is achieved and similar testing methodology is accepted only for some subclasses of global optimization problems, e.g. concave quadratic programming [21]. In case the methods are similar in basic ideas, structure, and intended application fields, competitive testing of such algorithms may be desirable. For example, competitive testing of interval methods [33], or of different implementations of evolution strategies [84] may be very informative. However, it would be hardly possible to design rational competitive testing experiment for algorithms from different classes.

A crucial obstacle for competitive testing of global optimization algorithms is the non-compatibility of different stopping conditions defining trade-off of two important criteria: fast finding of a good approximation and reliability. Only the algorithms with similar stopping conditions may be put into fair competing positions and tested for the collections of specified and randomly generated test functions. An example of competitive testing of different one-dimensional global optimization algorithms, but with compatible stopping conditions, is presented in [80]. However, definition of stopping conditions adequate to real world situations seems even more difficult than the development of a rational search algorithm. A common approach is to attack a problem using all available computing resources. The development of our algorithm was motivated by a quite frequent real life situation when an optimization problem should be solved using a customer's software for calculation of objective function values. Of course, the user would like to get global minimum precisely and with guarantee. However, normally he is not able to supply any information supporting such a claim. Using only the code, without a model of an objective function, the accuracy/reliability of a candidate solution can not be estimated. Definition of rational stopping condition in such a situation is hardly possible.

Algorithms for a general (not always precisely defined) class of smooth global optimization problems are based on different ideas [80]. The trade-off between fast finding of a good approximation and reliability is frequently based on heuristics, experience etc. If a new algorithm oriented to such a broad class of global optimization problems has no close prototypes, then competitive testing seems to be not realistic and a bit premature. The goal of **scientific** testing [34]: to show why and under what conditions the application of the proposed algorithm may be promising, what direction of development of the algorithm seems prospective.

The main criteria of efficiency of global optimization algorithms are the numbers of calls of the objective function and perhaps its gradient, and the calculation time of the optimization. The number of calls is useful when the objective function is expected to be "expensive", i.e. its calculation requires more time than the auxiliary calculations by the optimization algorithm. In case of the contrary relation, the calculation time is an important criterion. We use the numbers of function calls as efficiency criterion of sequential algorithms.

Criteria of efficiency of our algorithm are the number of objective function calls (Nrf) and the number of gradient calls (Nrg). If the gradient is expressed analytically, then a gradient call may cost the same as the objective function in some cases, in other cases up to *n* times a call of the objective function. If the gradient is evaluated using automatic differentiation then a gradient call usually costs about 3 times an objective function call. If the gradient is evaluated using finite difference approximations, then a call of the gradient function costs approximately *n*+1 calls of the objective function. In our experiments most gradients expressed analytically, otherwise (for instance in the two practical problems) they are found using finite difference approximations.

Interval methods are the closest prototypes of our algorithm, which may be considered as an extension of the former to the cases when interval arithmetic is not applicable. The results of a minimization of the same test functions by an available implementation, [43], of the interval method of [40] are presented for comparison. The method of [40] combines interval methods with real searches. The number of real function calls (Nrf) and the number of interval function calls (Nif) are the efficiency criteria of this algorithm. The authors of this algorithm state that the average cost of an interval function is twice the cost of a corresponding real function [40].

Comparison of two algorithms with respect to function calls of the overall optimization is well grounded when the stopping conditions are the same. Because the stopping conditions are different, we chose to use the numbers of calls needed to find the first and the last global minimizer, respectively, as the criterion of efficiency for both algorithms.

During the experimentation test functions defined by means of analytical formulas were used. An oracle for gradients was modeled by a subroutine approximating derivatives by forward differences. The testing methodology discussed in [79] requests to use test functions of different complexity (difficulty). Test functions with different dimensions and different numbers of local and global minimizers were used in the experiments. The dimensions and numbers of minimizers of the test functions are shown in Table 3. There the problems are divided into the categories: 1 local minimizer; 1 global minimizer and a few local; 1 global minimizer and many local; practical problems where the number of local minimizers is unknown; a few global minimizers.

Table 3. The dimensions and the numbers of local and global minimizers of test functions.

Function	*n*	Number of local min.	Number of global min.
Rosenbrock	2	1	1
McCormic	2	1	1
Box and Betts	3	1	1
Paviani	10	1	1
Generalized Rosenbrock	30	1	1
Goldstein and Price	2	4	1
Shekel 5	4	5	1
Shekel 7	4	7	1
Shekel 10	4	10	1
Levy 4	4	71000	1
Levy 5	5	10^5	1
Levy 6	6	10^6	1
Levy 7	7	10^8	1
Griewank	10	10^3	1
Cola	17	Unknown	1
Bone Growth	12	Unknown	Unknown
Six Hump Camel Back	2	6	2
Branin	2	23	5
Shubert	2	400	9
Hansen	2	760	9

Representatives of easy and moderately difficult test problems were chosen among well known and widely used functions. A detailed description of the functions can be found in [47], [48], C codes are available in [47]. Two practical problems were used as difficult test problems. The first practical problem, denoted by Cola, is the MDS problem which is discussed in [53]; the data used in this test correspond to the classical "Cola testing" problem. The number of variables is 17, the best currently known value of the objective function being 11.7464. There are many local minimizers with function values close to the global minimum [53]. The second practical problem denoted by Bone Growth is given in [2]. It is related to a linear growth model of the human mandible (the lower jaw). The dimension of this problem is 12, the best known function value being 205.104 and there are many local minimizers.

The default values of the parameters of the algorithms, which are typical for many applications, were used. In the proposed algorithm the local search is stopped when the gradient norm is less than 10^{-4}. Of course, for the analytically defined test functions much smaller tolerance may be fixed, but we wanted to model a real life situation with modest precision of input data; it seems not rational to request the precision of solution be higher than the precision of the data. For the parameter preventing multiple determination of the same local minimizer, $\varepsilon_{cluster}$, the two values 0.1 and 0.01 were chosen for all test examples. The tolerance μ of detecting whether a (known) global minimizer is considered to be global is 10^{-4}. The number of sample points was fixed equal to the empirically justified value $n_{sample}=2n+1$; such a choice has an advantage to perform testing in deterministic setting thus guaranteeing easy reproduction of testing results.

Table 4. The numbers of the objective (*Nrf*) and gradient (*Nrg*) functions calls needed to find the first and all minimizers with default parameters. $\varepsilon_{cluster}$=0.1. The two blocks of the table indicate whether there is one or several global minimizers.

Function	all minimizers	
	Nrf	Nrg
Rosenbrock	27	23
McCormic	11	10
Box and Betts	6	6
Paviani	14	11
Generalized Rosenbrock	359	346
Goldstein and price	75	51
Shekel 5	66	37
Shekel 7	24	13
Shekel 10	22	12
Levy 4	30502	23798
Levy 5	2786	2147
Levy 6	5292	4154
Levy 7	35	30
Griewank	124	81
Cola	8995	7226
Bone Growth	529	414

Function	first minimizer		all minimizers	
	Nrf	Nrg	Nrf	Nrg
Six Hump Camel Back	37	33	49	45
Branin	21	10	666	527
Shubert	26	15	6825	4412
Hansen	218	129	7655	5037

The numbers of the objective and gradient functions calls needed to find the first and all global minimizers with different values of $\varepsilon_{cluster}$ are given in Table 4 and Table 5. All global minimizers were found for all test problems. (For the BoneGrowth problem this means that the smallest known function value f^*=205.104 and the corresponding minimizer was found.)

Table 5. As Table 4 except that $\varepsilon_{cluster}$=0.01.

Function	all minimizers	
	Nrf	Nrg
Rosenbrock	27	23
McCormic	11	10
Box and Betts	6	6
Paviani	14	11
Generalized Rosenbrock	359	346
Goldstein and Price	81	56
Shekel 5	66	37
Shekel 7	24	13
Shekel 10	22	12
Levy 4	44316	33438
Levy 5	2823	2184
Levy 6	5389	4241
Levy 7	35	30
Griewank	124	81
Cola	9229	7413
Bone Growth	529	414

Function	first minimizer		all minimizers	
	Nrf	Nrg	Nrf	Nrg
Six Hump Camel Back	37	33	49	45
Branin	21	10	733	580
Shubert	26	15	8810	5380
Hansen	218	129	10296	6409

For many test functions the two choices of $\varepsilon_{cluster}$ give the same performance. For the 8 problems with differences, $\varepsilon_{cluster}$=0.1 is always the best, however the differences are not very large (except for the Levy 4 function). In general we conclude that the experiments indicate some independence of the choice of $\varepsilon_{cluster}$.

The algorithm has difficulties with functions having very many oscillations and local minima, like the Levy, Shubert and Hansen functions. For such functions the performance seems to be a bit chaotic. For Levy 4, which is the "easiest" of the four

Levy functions, the solution requires many function evaluations, whereas the most difficult one, Levy 7, is solved rather quickly because the algorithm accidentally finds the global minimizer at the beginning of the optimization. The performance in such cases may depend on how close the starting points of the local searches are to a global optimizer. Thus the performance may change if the boxes *D* of feasible regions are shifted. For other difficult functions with many local minimizers, like the Griewank, Cola and BoneGrowth functions, the algorithm works rather well.

Table 6. Results of interval algorithm [43]. The number of subdivisions per iteration, n_d, is 2 for all problems. *Nrf* is the number of real function calls, *Nif* is the number of interval function calls. The two blocks of the table indicate whether there is one or several global minimizers.

Function	All minimizers	
	Nrf	Nif
Rosenbrock	170	13
McCormic	97	9
Box and Betts	97	21
Paviani	366	41
Generalized Rosenbrock	6513	125
Goldstein and Price	129	9
Shekel 5	172	17
Shekel 7	166	17
Shekel 10	211	17
Levy 4	614	6839
Levy 5	555	202
Levy 6	527	330
Levy 7	543	506
Griewank	fails	
Cola	fails	
Bone Growth	fails	

Function	first minimizer		all minimizers	
	Nrf	Nif	Nrf	Nif
Six Hump Camel Back	225	554	321	819
Branin	324	119	fails	
Shubert	524	1543	fails	
Hansen	89	17	862	1421

The interval algorithm [43] used for comparison is based on a combination of local searches, branch-and-bound technique and interval arithmetic. We have used different values of the parameter n_d which determines how many subboxes a box is split into at each iteration: n_d = 2 (which is the default value used for the corresponding parameter of our method), n_d = n+1 and n_d = *a value tuned for each individual test problem*. The numbers of real and interval function calculations needed to find the first and all minimizers are shown in Table 6, Table 7 and Table 8.

Table 7. As Table 6 except that n_d is n+1.

Function	all minimizers	
	Nrf	Nif
Rosenbrock	104	19
McCormic	96	13
Box and Betts	103	33
Paviani	524	221
Generalized Rosenbrock	14260	1921
Goldstein and Price	243	226
Shekel 5	98	41
Shekel 7	101	41
Shekel 10	101	41
Levy 4	111	41
Levy 5	127	61
Levy 6	235	85
Levy 7	235	113
Griewank	186	221
Cola	fails	
Bone Growth	fails	

Function	first minimizer		all minimizers	
	Nrf	Nif	Nrf	Nif
Six Hump Camel Back	268	4532	342	5411
Branin	506	453	803	687
Shubert	940	3802	1550	4776
Hansen	193	200	892	1686

The tables demonstrate some dependence of the choice of n_d. Table 8 shows that sometimes quite a lot may be gained by tuning this parameter. In the Generalized Rosenbrock, for instance, the number of the real function calls varies from 2698 to 14260.

The algorithm failed to minimize the two practical problems Cola and Bone Growth. To be more specific the computation broke down because of overflow of memory. For the Cola problem the number of unexplored boxes generated by the algorithm was more than 131072 at the stopping time, and the number of interval function calls was greater than 393213. Similar results were seen for the Bone Growth problem.

Table 8. As Table 6 except that n_d is tuned for each individual problem.

Function	n_d	All minimizers	
		Nrf	Nif
Rosenbrock	3	104	19
McCormic	2	97	9
Box and Betts	1	93	11
Paviani	2	366	41
Generalized Rosenbrock	7	2698	431
Goldstein and Price	1	120	5
Shekel 5	4	94	33
Shekel 7	4	85	33
Shekel 10	4	85	33
Levy 4	5	111	41
Levy 5	6	127	61
Levy 6	4	178	49
Levy 7	4	233	57
Griewank	8	186	221
Cola	any	fails	
Bone Growth	any	fails	

Function	n_d	first minimizer		all minimizers	
		Nrf	Nif	Nrf	Nif
Six Hump Camel Back	1	273	352	365	443
Branin	3	506	453	803	687
Shubert	4	357	1008	1060	3472
Hansen	2	89	17	862	1421

The numbers of calls needed to minimize the mathematical test problems are relatively small, i.e. the method performs well for these problems.

The comparison of performance of both algorithms is summarized in Table 9. For our algorithm $\varepsilon_{cluster}$=0.1 was chosen. For the interval algorithm the number of subdivisions in each iteration was chosen equal to n+1, which is generally the most efficient default value for the problems we tested. The numbers of the functions calls before all global minimizers are detected is used as criteria. In most cases all the global minimizers (or the best currently known values) were found by our algorithm in the first inner loop of the algorithm, i.e. the garbage set G was not necessary in these tests. The interval method "fails" for both practical problems.

Table 9. Numbers of function and gradient or interval calls needed to find all global minimizers. The columns with “Black box” correspond to the proposed method, the column "Interval" corresponds to [43]. The notations: Nrf – number of real function calls, Nrg – number of real gradient calls, Nif – number of interval function calls.

Function	Black box		Interval	
	Nrf	Nrg	Nrf	Nif
Rosenbrock	27	23	104	19
McCormic	11	10	96	13
Box and Betts	6	6	103	33
Paviani	14	11	524	221
Generalized Rosenbrock	359	346	14260	1921
Goldstein and Price	75	51	243	226
Shekel 5	66	37	98	41
Shekel 7	24	13	101	41
Shekel 10	22	12	101	41
Levy4	30502	23798	111	41
Levy5	2786	2147	127	61
Levy6	5292	4154	235	85
Levy7	35	30	235	113
Griewank	124	81	186	221
Cola	8995	7226	fails	
Bone Growth	529	414	fails	
Six Hump Camel back	49	45	342	5411
Branin	666	527	803	687
Shubert	6825	4412	1550	4776
Hansen	7655	5037	892	1686

The test results indicate that the proposed algorithm is stable in the sense that it has solved all the problems we have tried. The use of two values of the parameter for preventing multiple determination of the same local minimizer, $\varepsilon_{cluster}$ indicates a relative robustness with respect to this parameter. Although our algorithm is rather efficient for many problems, we occasionally see slow convergence for problems with very many local minimizers, e.g. the Levy problems.

It is important to notice that interval methods are very efficient for test functions composed by not too large number of trigonometric functions (like the Shubert and Levy problems). This may be explained by availability of efficient methods for calculating interval inclusion functions in such cases, thus making the threshold test

(4) efficient. The heuristic lower bounds found by our method, however, are often much more pessimistic since we cannot utilize inequalities such as $|\sin(x)| \le 1$.

In conclusion, the new method has been solving all the problems we have tried. It is sometimes even faster than the interval method, but for some problems it is much slower. On the other hand it seems to be generally more applicable. The proposed method can not guarantee evaluation of global minimum with prescribed accuracy, but it seems a good candidate to substitute the interval methods when the latter are not applicable, e.g. in a "black box" situation.

3.4 Comparison with Lipschitz algorithm

Comparison of two different algorithms with different stopping conditions is difficult. The test functions for unconstrained global optimization (whose global minimizers are not on the edge of the feasible region) from [31] were optimized using black box global optimization algorithm and Lipschitz global optimization algorithm with simplicial partitioning. The results are given in Table 10. The gradients of functions were not provided for the black box algorithm, they were estimated using finite differences. The numbers of objective function calls needed to find the first and the last global minimizers are given as results of the black box global optimization algorithm. The numbers of objective function calls of overall optimization are given as results of the Lipschitz global optimization algorithm with simplicial partitioning.

Table 10. Comparison of black box and Lipschitz global optimization algorithms.

Test function	Black box	Lipschitz
3	32 1287	2185
3.1	38 1607	1559
3.2	57 1307	1225
3.3	32 1644	1429
4	413	70
7	110	3117
9	228	3773
9.1	297	3890
9.2	259	3729
10	64	848
11	20	1566
12	28	4001
13	131034	4084

The table does not help to compare algorithms because the stopping conditions, input and results of algorithms are different. The black box global optimization algorithm stops after defined duration of time. The numbers of objective function calls needed to find the first and the last global minimizer is the criterion of efficiency of algorithm. All found global minimizers are outputted. Minimizers are found during local searches whose are stopped when the gradient norm is less than 10^{-4}.

Lipschitz global optimization algorithm with simplicial partitioning guarantees that the global minimum with predefined accuracy is found. The Lipschtiz constants of objective functions should be provided. The number of objective function calls of overall optimization is the criterion of efficiency of algorithm. Only one point is outputted – the point of the best value of the objective function. The dimensionality of successful Lipschitz optimization applications is modest.

The black box global optimization is generally more applicable. The algorithm can not guarantee evaluation of global minimum with prescribed accuracy, but Lipschitz constants of functions are not needed.

3.5 Parallel algorithm

The most straightforward implementation of a parallel algorithm of the method is using geometric parallelization. In this case the domain of the problem is initially divided into sub-problems whose are distributed to parallel processors. Then processors work independently running Algorithm 8 and do not exchange any latter generated sub-problems. The currently known best value of the objective function $\overline{f}$ can be exchanging between processors. Such an algorithm corresponds to the distributed paradigm of parallel programming with the static load balance strategy. The termination of the algorithm is the same as the stopping condition in the sequential case – time limit. The existence of the outer loop in the Algorithm 8 ensures that all processors always have work to do, but it is difficult to know do processors perform computations on equally promising sub-problems.

The portable parallel algorithm has been implemented in C++ and MPI (Message-Passing Interface – a standard specification for message-passing libraries). Portability problems of incompatible C++ translators and MPI implementations had to be overcome. The latest implementation is available for noncommercial use at [88]. The algorithm has been tested on the available parallel systems at Edinburgh Parallel Computing Center:

- Sun HPC 6500 UltraSPARC-II based system with Sun MPI,
- Beowulf-type PC cluster with MPICH 1.2.0,
- Cray T3E,

and IBM RS/6000 SP at Vilnius Gediminas Technical University.

The usually used criteria of efficiency of parallel algorithms – the speedup s_m and the efficiency e_m, are not suitable for evaluation of parallel implementations of the proposed method, because the stopping condition is time limit and s_m=1 and $e_m = \frac{1}{m}$.

The pseudo efficiency pe_m and the proportion of time spent on calculations (other time is spent on communications or being idle) are also not well suitable in our case. They are maximal when there are no communications at all and therefore they suggest not exchange the currently known best value of the objective function $\overline{f}$, what is not desirable practically.

When the stopping condition is time limit, the evaluation of criteria of parallel algorithm is difficult. The stopping condition must be changed to enable evaluation of criteria. The stopping condition could be the predefined number of outer iterations. The experiments with the sequential version have shown that all the global minimizers were found in the first inner loop of the algorithm.

Typical efficiency criteria as functions of the number of processors are shown in Figure 14. These figures are obtained running the parallel algorithm of the method on the Beowulf-type PC cluster when the stopping condition is one outer iteration. Initially, the domain of the problem is subdivided into sub-problems so that each processor gets one sub-problem. Then processors exchange the currently known best value of the objective function, but do not exchange any latter generated sub-problems.

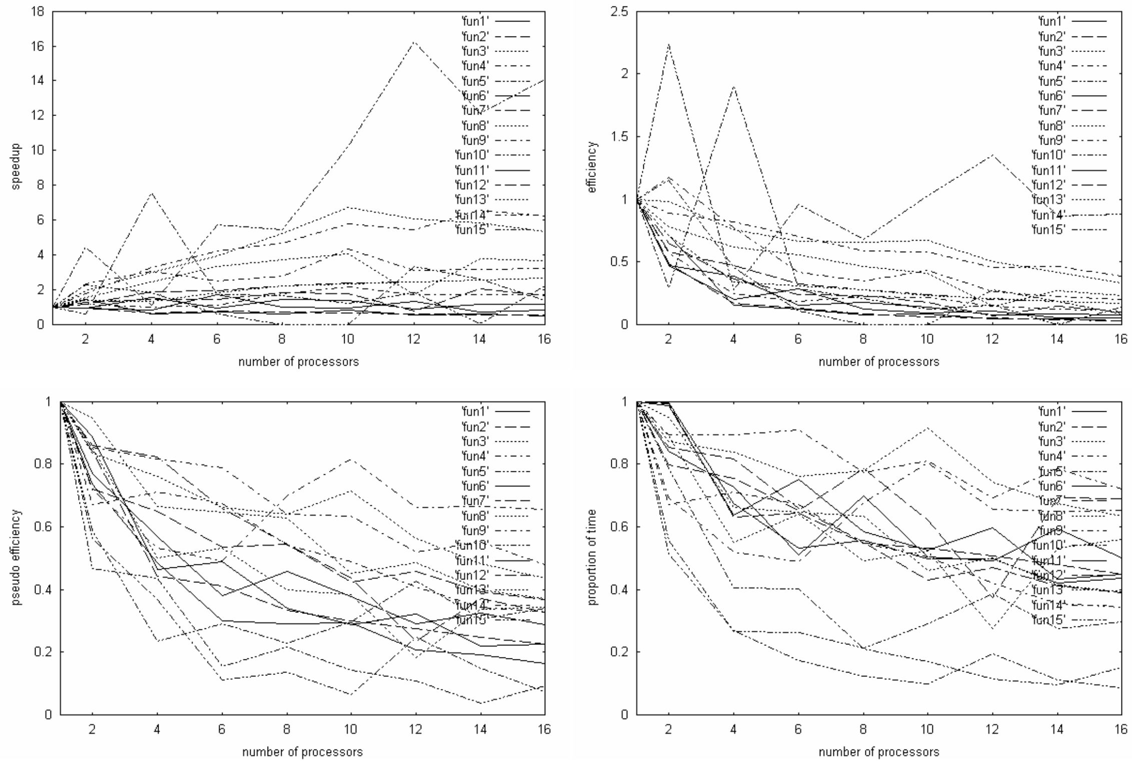


Figure 14. Speedup, efficiency, pseudo efficiency and proportion of time spent on calculations as functions of the number of processors.

We use as a measure of the amount of work the total number of calls of the objective function. This measure is good when the objective function is expected to be "expensive", i.e. its calculation requires more time than the auxiliary calculations by the optimization algorithm, or when the amount of the auxiliary calculations is proportional to the number of calls of the objective function.

Figures show that anomalies take place, and therefore the speedup and the efficiency are not appropriate criteria. The pseudo efficiency and the proportion of time spent on calculations show similar results. The pseudo efficiency is always less than 1. This shows that the measure of amount of work is chosen properly.

The worst case pseudo efficiency and processor utilization are low. However the average pseudo efficiency and processor utilization are not very bad. The work load is not always balanced good. Possibly the load balancing could be improved using dynamic load balancing, but it is not always the case as it was shown in section 2.5.4.

Moreover these results are gotten when the stopping condition is one outer iteration and do not represent the method when the stopping condition is time limit.

3.6 Conclusions

The black box global optimization algorithm inspired by interval methods has been proposed. The sequential algorithm of the method has been implemented in C++, and the numerical experiments with some well-known test problems as well as with two practical problems have been performed. The experimental testing shows that the method is applicable and performs well not only for the standard test functions but also for practical problems.

The portable parallel algorithm of the method has been implemented in C++ and MPI. The algorithm has been tested on the various parallel systems. The criteria of the parallel algorithm have been discussed.

4 Application of black box global optimization algorithm for practical problems

In this chapter application of the presented black box global optimization method is presented. The method is applied to practical problems: multidimensional scaling, growth model of the human mandible and many body problems. The results of optimization are discussed.

The application of the presented black box global optimization algorithm depends on the following properties of the algorithm:

- It is multidimensional global optimization algorithm.
- Feasible region is box constrained. The algorithm can be applied for unconstrained global optimization problems.
- The algorithm is applicable in a “black box” situation. It does not depend on how the objective function is expressed.
- The trust region local search algorithm with BFGS is used, gradients are calculated using provided function or estimated using forward differences.

4.1 Multidimensional scaling

Experimental sciences collect large amounts of data supposed for analysis by different techniques. Frequently researcher’s experience and intuition should be combined with mathematical methods to elicit the desired knowledge from the data. However, human heuristic abilities are mostly developed and oriented to patterns in three (or two) dimensional space. A mapping of multidimensional data into the patterns of two- or three- dimensional space may be greatly aid the heuristic analysis and enhance the validity of the conclusions. Multidimensional scaling is a widely used technique to analyze structure and inherent dimensionality of sets of multidimensional data a well as visualization of data by means of their mapping into spaces of low dimensionality [16], [6], [25].

Multidimensional scaling addresses the problem how N objects represented by proximity data can be faithfully visualized as points in a low-dimensional Euclidean space [42]. Proximity data are represented as pairwise dissimilarity values. The dissimilarity of i^{th} object to j^{th} object is defined as a real number δ_{ij}. A spatial representation of the objects should be determined. Objects are represented as points in m-dimensional space. The distance between two points $\mathbf{x}_i$ and $\mathbf{x}_j$ is usually measured by the Euclidean distance $d_{ij}=\|\mathbf{x}_i-\mathbf{x}_j\|$. Sometimes the raw dissimilarity data are not suitable for Euclidean embedding and dissimilarities are transformed to disparities $D_{ij}=D(\delta_{ij})$.

The quality of visualization is measured by a *stress* function aggregating the differences between the proximity values and the distances between the respective points. Multidimensional scaling problem could be solved optimizing the stress function. The coordinates of the N points in m-dimensional space should be found such that the stress function would be minimal. The most frequently used stress function is

$$f(X)=\sum_i\sum_j w_{ij}(d_{ij}-D_{ij})^2 ,$$

where w_{ij} are nonnegative weights. They may be used to normalize the values of the disparities. A local, global or intermediate normalization is defined as follows in [42]:

$$w_{ij}^{(l)}=\frac{1}{N(N-1)D_{ij}^2} ,$$

$$w_{ij}^{(g)}=\frac{1}{\sum_l\sum_m D_{lm}^2} ,$$

$$w_{ij}^{(m)}=\frac{1}{D_{ij}\sum_l\sum_m D_{lm}} .$$

The different choices correspond to a minimization of relative, absolute or intermediate error. The weightings might also be used to discount disparities with a high degree of experimental uncertainty.

Sometimes the squared Euclidean distances are used to simplify computations:

$$f(X)=\sum_i\sum_j w_{ij}(\|\mathbf{x}_i-\mathbf{x}_j\|^2-\delta_{ij}^2)^2 .$$

Such function is called SSTRESS. A more natural choice of stress function is

$$f(X)=\sum_i\sum_j w_{ij}(\|\mathbf{x}_i-\mathbf{x}_j\|-\delta_{ij})^2 .$$

In most cases symmetric dissimilarities are used: $\delta_{ij}= \delta_{ji}$ what allows to reduce computations:

$$f(X)=\sum_{j<i} w_{ij}(\|\mathbf{x}_i-\mathbf{x}_j\|-\delta_{ij})^2 .$$

If weighting is not used the function may be further reduced:

$$f(X)=\sum_{j<i}\left(\sqrt{\sum_{k=1}^{m}\left(x_{i,k}-x_{j,k}\right)^2}-\delta_{i,j}\right)^2 ,$$

where $x_{i,1..m}$ are the coordinates of the i^{th} point, i=1..N, j=1..N.

Minimization of stress function is difficult because of the following features:

- the practical minimization problems normally are high dimensional, number of optimization variables is equal to $N\times m$,
- the stress function is not everywhere differentiable,
- the problem is multimodal,
- the stress function is invariant with respect to translation, rotation and mirroring.

Some of the mentioned difficulties may be, at least partially, avoided. For example, the subsets of the local minimizers may be reduced to the points [42] by means of fixing of several coordinates:

$x_{i,i..m}$=0, for i=1..m

and defining some to be non negative:

$x_{i,j-1}\geq 0$, for i=2..m+1.

Non fixed coordinates are arguments of the problem:

$x_{2,1}=u_1,\ldots,x_{p,m}=u_n$.

The dimensionality of the problem is:

$$n=\frac{m(2N-m-1)}{2} .$$

Many authors minimize different versions of stress function by means of either local techniques or their modifications. Some modifications of local methods are derived to achieve non-local behavior of the method. Most widely known method of such a kind is SMACOF based on majorization of an objective function [17], [18]. Although the method not necessary converges to the closest local minimizer, its search strategy is rather local.

The tunneling method is known as one of global optimization methods [80]. It was adopted to the problems of multidimensional scaling and investigated in detail in the doctoral dissertation by P.Groenen [26]. It was very efficient while used by such an experienced researcher of multidimensional scaling as P.Groenen. However, the average user would hardly use this method because of its dependence on many heuristically tuned parameters.

Although improved local search procedures are used for some applications of multidimensional scaling certain applications can be solved only with global optimization. Two examples of such applications are described in [52]. One of applications is the estimation of the GSM mobile’s position using the measured powers of the 6 signals received from surrounding base stations.

A frequently used test problem for multidimensional scaling algorithms [52], [53], [25] is based on experimental results on testing of several soft drinks [25]. 38 students have tested ten different Cola brands. Each pair was judged on its dissimilarity on a 9 point scale (1 – very similar, 9 – completely different). The accumulated dissimilarities are the data of the problem. The scaled down accumulated dissimilarities are given in Table 11.

Table 11. Scaled accumulated dissimilarities between 10 Cola brands.

	Pepsi	Coke	Classic Coke	Diet Pepsi	Diet Slice	Diet 7-Up	Dr. Pepper	Slice	7-Up	Tab
Pepsi	…									
Coke	1.27	…								
Classic Coke	1.69	1.43	…							
Diet Pepsi	2.04	2.35	2.43	…						
Diet Slice	3.09	3.18	3.26	2.85	…					
Diet 7-Up	3.20	3.22	3.27	2.88	1.55	…				
Dr. Pepper	2.86	2.56	2.58	2.59	3.12	3.06	…			
Slice	3.17	3.18	3.18	3.12	1.31	1.64	3.00	…		
7-Up	3.21	3.18	3.18	3.17	1.70	1.36	2.95	1.32	…	
Tab	2.38	2.31	2.42	1.94	2.85	2.81	2.56	2.91	2.97	…

The goal of this multidimensional scaling problem is to find the configuration of 10 objects representing each Cola brand in two-dimensional space, which would help to interpret the data. It is shown in [52] that there are many local minima and interpreting the data on the basis of the achieved configuration from local minima leads to different results. So it is necessary to find the global minimum and the corresponding configuration which explains the data best.

The number of objects for this global optimization problem N=10. The configuration of objects in m=2 dimensional space should be find. The number of variables n=17. The feasible region is D=([0,4]^2,[-4,4]$^{n-2}$). The objective function of the problem is

$$f(X) = \sum_{j<i} \left(\sqrt{\sum_{k=1}^{m} \left(x_{i,k} - x_{j,k} \right)^2} - \delta_{i,j} \right)^2 .$$

There are many local minimizers with function values close to the global minimum. The best known value of the objective function f^*=11.7464. The black box global optimization algorithm has found this global minimum after 25242 calls of the objective function. The found global minimizer is

U^*=(0.6519, 1.3019, 0.0992, -0.8838, -0.8796, 0.2047, -3.2841, 0.8512, -3.4625, 2.5325, -0.8952, 1.4099, -3.0737, 1.9626, -2.9787, -0.8078, -1.6898).

The representing solution configuration of the cola problem is shown in Figure 15.

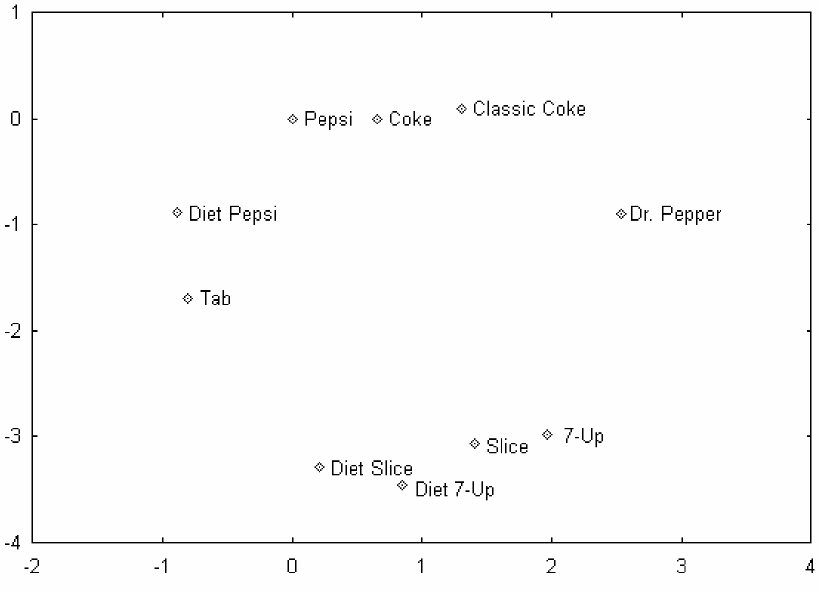
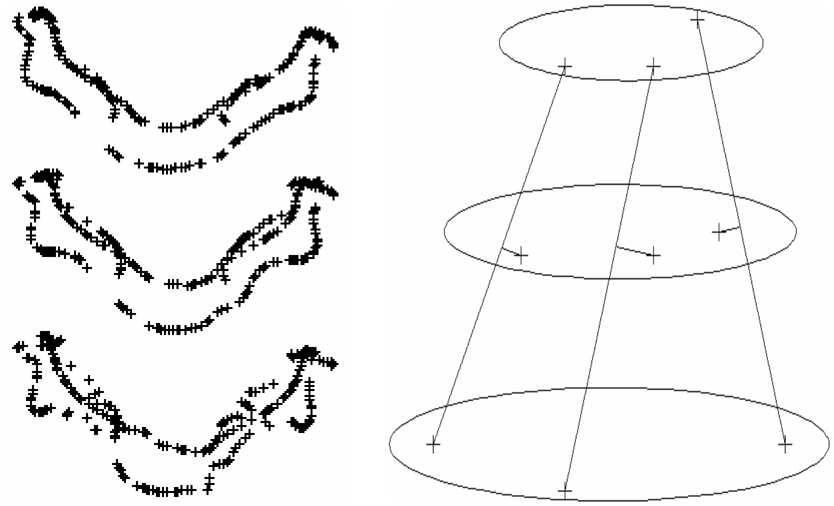


Figure 15. The solution of the Cola problem.

An available implementation [43], of the interval method [40], which was used for the comparison in section 3.3 has not found the global minimum of the problem and has broken down because of overflow of memory.

4.2 Growth model of the human mandible

The problem represents a growth model of the human mandible (the lower jaw). The data of the problem are the coordinates of 271 points of equivalent morphology from the 3 mandibles of the same patient at the age of 9 months, 21 month and 7 years [2]. The goal of the problem is to position the three mandibles in the space so that the sum of distances of points from the middle mandible to the lines connecting corresponding points from the first and third mandibles is minimal (Figure 16). The middle mandible is fixed and each of the two others has 6 degrees of freedom: 3 angles of rotation and 3 directions of translation. Thus the dimension of the problem is n=12. The domain D=([-π, π]^6, [-120,120]^6).



$$f(X)=\sum_{i=1}^{271} h_i$$

Figure 16. The growth model problem.

The distance of the point from the middle mandible to the line connecting corresponding points from the first and third mandibles is found using the expression of area of triangle:

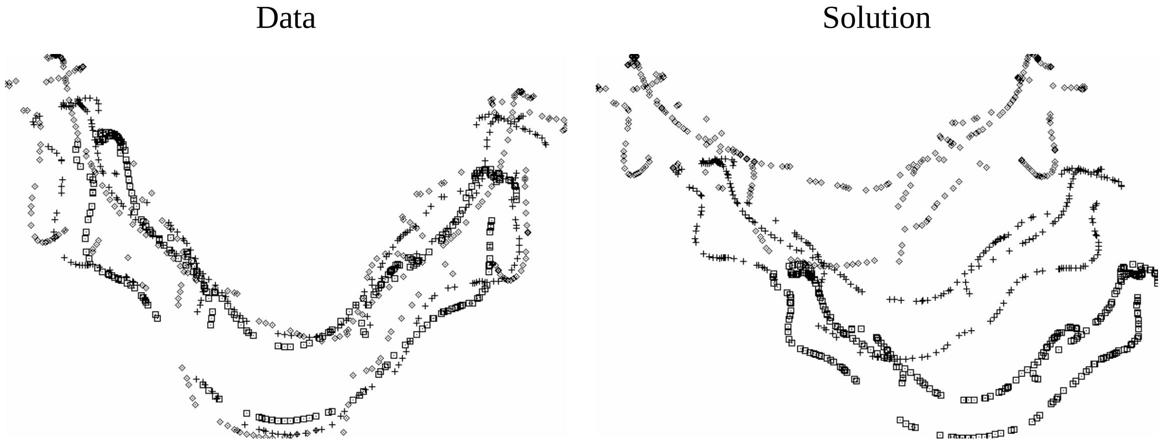
$$h=\frac{2S}{b},$$

where *S* is area of the triangle whose corners are the points of equivalent morphology from the three mandibles, *b* is the distance between the points from the first and third mandibles. Area of the triangle is:

$$S = \sqrt{p(p-a)(p-b)(p-c)}\ ,\ \ p = \frac{1}{2}(a+b+c)\ ,$$

where *a* is the distance between the points from the first and second mandibles and *c* is the distance between the points from the second and third mandibles.

The best known function value of the problem is f^*=205.104. The black box global optimization algorithm has found the global minimum after 13101 calls of the objective function. The found minimizer is
X^*=(-0.125288, -0.048084, 0.0683822, -5.28448, 13.5913, 47.4381, 0.100655, 0.00663149, 0.0861344, 15.1711, -19.2757, -44.7489).
The corresponding solution of the growth problem is shown in Figure 17.



f=610.342

f^*=205.104

Figure 17. The solution of the growth problem.

An available implementation [43], of the interval method [40], which was used for the comparison in section 3.3 has not found the global minimum of the problem and has broken down because of overflow of memory.

4.3 Many-Body Problem

There are many problems in physics, chemistry and material science whose require to know the total energy of a system of atoms as a function of atomic positions.

Structural and energetic properties of complex systems could be calculated using empirical interatomic potentials. A new approach for constructing potentials incorporating the dependence of bond orders is presented in [74]. An environment dependent bond order is included into the potential because it depends on local geometry and plays the crucial role. The interatomic potential is taken to have the form

$$E = \sum_i E_i = \frac{1}{2}\sum_{i \neq j} V_{ij} \ ,$$

$$V_{ij} = f_C(r_{ij})[a_{ij} f_R(r_{ij}) + b_{ij} f_A(r_{ij})],$$

where E is the total energy of the system which is decomposed into site energies E_i and bond energies V_{ij}. The indices i and j run over all atoms of the system, r_{ij} is the distance between i^{th} and j^{th} atoms.

The term f_R represents a repulsive pair potential which includes the orthogonalization energy when atomic wave functions overlap. The term f_A represents an attractive pair potential associated with bonding. These terms are expressed using exponential functions:

$$f_R(r) = A\exp(-\lambda_1 r),$$

$$f_A(r) = -B\exp(-\lambda_2 r).$$

The term f_C is a smooth cutoff function to limit the range of potential:

$$f_C(r) = \begin{cases} 1, r \leq R - D \\ \frac{1}{2} - \frac{1}{2}\sin\left[\frac{\pi(r-R)}{2D}\right], R - D < r < R + D \\ 0, r \geq R + D \end{cases}$$

which goes from 1 to 0 in a small range around R. For most structures R is chosen to include only the first neighbor shell.

The function b_{ij} represents a measure of the bond order and is assumed to be a monotonically decreasing function of the coordination of i^{th} and j^{th} atoms. It also includes terms whose act to limit the range of interaction to the first neighbor shell:

$$b_{ij} = \left(1 + \beta^n \xi_{ij}^n\right)^{-\frac{1}{2n}},$$

$$\xi_{ij} = \sum_{k \neq i, j} f_C(r_{ik}) g(\theta_{ijk}) \exp[\lambda_3^3 (r_{ij} - r_{ik})^3],$$

$$g(\theta) = 1 + \frac{c^2}{d^2} - \frac{c^2}{d^2 + (h - \cos\theta)^2},$$

where θ_{ijk} is the bond angle between bonds *ij* and *ik*.

The cosine of the bond angle could be found as:

$$\cos\theta_{ijk} = \frac{(x_{1j} - x_{1i})(x_{1k} - x_{1i}) + (x_{2j} - x_{2i})(x_{2k} - x_{2i}) + (x_{3j} - x_{3i})(x_{3k} - x_{3i})}{r_{ij} r_{ik}},$$

where x_{ik} is the *i*th coordinate of *k*th particle.

The function a_{ij} consists of range limiting terms:

$$a_{ij} = \left(1 + \alpha^n \eta_{ij}^n\right)^{-\frac{1}{2n}},$$

$$\eta_{ij} = \sum_{k \neq i, j} f_C(r_{ik}) \exp[\lambda_3^3 (r_{ij} - r_{ik})^3].$$

A, B, λ_1, λ_2, α, β, n, c, d, h, λ_3, R, D are fitted parameters. Parameters for arsenic potential denoted by As and for two silicon potentials denoted by Si(B) and Si(C) are given in Table 12. These parameters are taken from [1], [74] and [75]. Parameters for other potentials are discussed in [5].

Table 12. The parameters of the potentials.

	As	Si(B)	Si(C)
A	10.45561332	3.2647e+3	1.8308e+3
B	14.41961332	9.5373e+1	4.7118e+2
λ_1	6.739581257	8.7963	6.7339
λ_2	4.886847795	3.6001	4.7036
α	0.0	0.0	0.0
β	0.00748809	0.33675	1.0999e-6
n	0.60879133	22.956	0.78734
c	5.2731318	4.8381	1.0039e+5
d	0.75102662	2.0417	16.216
h	0.15292354	0.0000	-0.59826
λ_3	0.0	0.0	0.0
R	1.2381	1.1048	1.0496
D	0.0503062	0.073654	0.055240

The goal of the problem presented in [1] is to find the minimum energy of clusters of particles whose interact through well defined many-body interaction potentials. The energy of a particle depends on the distances and angles with respect to other particles, different particles have different energies.

The objective function *f* is a function of the atomic positions of the particles. When minimizing the objective function, we fix the first particle at the origin, lay the second particle on the positive *x*-axis and the third particle on the *xy*-plane. The position of the third particle is defined using the distance from the origin and the polar angle. The positions of additional particles are defined as the coordinates in three-dimensional space.

The physics involved in the problems imposes restrictions on the variables. The distances between the first-second and first-third particles are taken to lie in [0,1.30] for As and [0,1.16] for Si. The polar angle is taken to lie in $[0,\pi]$. The coordinate variables are in [-1.5,1.5], however the third coordinates of the fourth and fifth particles constrained to be nonnegative and nonpositive respectively. The restrictions on the variables of the problems are given in Table 13.

Table 13. The restrictions on the variables of the many-body problems.

Particle	As	Si
2nd	$0 \le x_1 \le 1.30$	$0 \le x_1 \le 1.16$
3rd	$0 \le x_2 \le 1.30$, $0 \le x_3 \le \pi$	$0 \le x_2 \le 1.16$, $0 \le x_3 \le \pi$
4th	$-1.5 \le x_4, x_5 \le 1.5$, $0 \le x_6 \le 1.5$	$-1.5 \le x_4, x_5 \le 1.5$, $0 \le x_6 \le 1.5$
5th	$-1.5 \le x_7, x_8 \le 1.5$, $-1.5 \le x_9 \le 0$	$-1.5 \le x_7, x_8 \le 1.5$, $-1.5 \le x_9 \le 0$
6th	$-1.5 \le x_i \le 1.5$, i=10,11,12	$-1.5 \le x_i \le 1.5$, i=10,11,12

The problems are difficult because of large number of local minimizers. The authors of [1] extracted some characteristics of the problems performing local searches from 10000 random points within the feasible regions. This gave the probabilities that a local search started from a random point would reach the global minimum. Based on the estimated probabilities the authors conclude that all problems with 3 and 4 particles and Si(B) with 5 particles are easy to solve (the estimated probabilities are larger than 4%). The others are more difficult, Si(C) with 5 and 6 particles already respectively difficult (the estimated probabilities are less than 0.01%). Problems with larger number of particles are increasingly difficult to solve.

The results of minimization the many-body problems using the black box global optimization algorithm: the smallest found value of the objective function and the number of the objective function calls needed to find the global minimum if it is found, are given in Table 14. When the global minimum is the same as in [1], it is shown in brackets. The found global minimums for As problems are less than given in [1], where the details about global minimizers are not given. It is possible that the parameters of the As problems are wrong, so it is not recommended to use the results of As problems for comparison of performances of the algorithms.

Table 14. The results of minimization.

Number of particles	As	Si(B)	Si(C)
3	-8.07739 182	[-7.87096] 64	[-5.33194] 88
4	-12.8546 45498	[-15.7057] 186	[-7.99786] 1631
5	-16.6568 11808	[-20.3985] 327	-10.2009
6	-16.816	[-26.5196] 256460	-13.9692

The global minimum of Si(B) problem with 3 particles is f^*=-7.87096. The found corresponding solution is X^*=(0.852, 0.852, 1.047) which represents the particles on the corners of a regular triangle as in Figure 18.

The global minimum of Si(B) problem with 4 particles is f^*=-15.7057. The found corresponding solution is X^*=(0.852, 0.852, 1.047, 0.426, 0.246, 0.696) which represents the particles on the corners of a regular tetrahedron as in Figure 18.

The global minimum of Si(B) problem with 5 particles is f^*=-20.3985. Two found corresponding symmetric solutions are X^*=(0.871, 0.871, 1.047, 0.436, 0.251, 0.706, 0.436, 0.252, -0.706) and X^*=(0.867, 0.867, 1.903, 0.438, 0.615, 0.436, 0.438, 0.615, -0.436). The solutions represent the particles on the corners of two regular tetrahedrons as in Figure 18.

The global minimum of Si(B) problem with 6 particles is f^*=-26.5196. The same solution was found two times with exchanged fourth and sixth particles: X^*=(0.871, 0.871, 1.57, 0.436, 0.436, 0.616, 0.436, 0.436, -0.616, 0.871, 0.871, 0.0) and X^*=(0.871, 0.871, 1.57, 0.871, 0.871, 0.0, 0.436, 0.436, -0.616, 0.436, 0.436, 0.616). The solution represents the particles on the corners of a regular octahedron as in Figure 18.

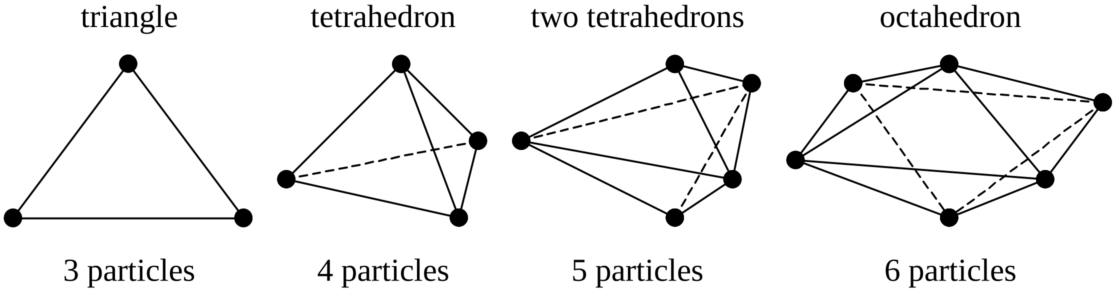


Figure 18. Solutions of Si(B) problems.

In [1] the results of minimization the As and Si(B) problems using eight stochastic algorithms are given. The performances of the black box global optimization algorithm and these eight stochastic algorithms were compared using results of Si(B) problem. The results of minimization Si(B) problem using the stochastic algorithms (CSR2, CRS3, CRS4, CRS5, ABSA, SA, TMSL, MSL) and the black box global optimization algorithm (BB) are given in Table 15.

Table 15. Results of stochastic algorithms [1] and black box algorithm.

MBP		CRS2	CRS3	CRS4	CRS5	ABSA	SA	TMSL	MSL	BB
3	f^*	[-7.87]	[-7.87]	[-7.87]	[-7.87]	[-7.87]	[-7.87]	[-7.87]	[-7.87]	[-7.87]
	FE	1123	1094	755	808	13992	13978	212	210	64
4	f^*	[-15.71]	[-15.71]	[-15.71]	[-15.71]	[-15.71]	[-15.71]	[-15.71]	[-15.71]	[-15.71]
	FE	4304	3724	2418	2967	27988	46458	2520	5312	186
5	f^*	[-20.40]	-20.31	[-20.40]	-15.70	-20.31	-20.31	-18.97	[-20.40]	[-20.40]
	FE	11111	18348	3971	4321	36417	112951	3594	12954	327
6	f^*	-24.51	-24.51	[-26.52]	-24.51	-25.98	-23.12	-24.44	-24.51	[-26.52]
	FE	37677	64067	44980	42134	151109	206166	4139	18666	256460

All algorithms have found the global minimum of the problems with 3 and 4 particles and the efforts needed to find the global minimum were used for comparing as in [1].

The black box global optimization algorithm is the best one, because it requires the smallest number of the objective function evaluations to find the global minimum.

Not all stochastic algorithms have found the global minimum of the problems with 5 and 6 particles and the reliabilities in finding the best solution were used for comparing as in [1]. Only one stochastic algorithm has solved Si(B) problem with 6 particles and only three of them have solved the problem with 5 particles. Again the black box global optimization algorithm is one of the best, because it has solved both Si(B) problems with 5 and 6 particles. The algorithm is both reliable and efficient. In [1] the reliability rankings were roughly negatively correlated with the efficiency rankings for all except two algorithms whose were neither reliable nor efficient.

Results

1. The general n-dimensional simplex-based branch and bound algorithm for Lipschitz optimization has been proposed. The rules of selection, covering, branching and bounding have been justified by results of experimental investigations. The performance of the algorithm, using the number of function evaluations required to solve the problem, has been evaluated and compared with other branch and bound algorithms for Lipschitz optimization.
2. The parallel master-slave and distributed simplex-based branch and bound global optimization algorithms with static and dynamic load balancing have been implemented using C++ and MPI. The criteria of parallel algorithms have been evaluated and the superiority of distributed algorithms has been shown.
3. The black box global optimization algorithm inspired by interval methods is proposed. The sequential algorithm of the method has been implemented and tested using the numerical experiments with some well-known test problems as well as with two practical problems. The experimental testing has shown that the method is applicable and performs well not only for the mathematical test functions but also for the practical problems.
4. The portable parallel algorithm of the black box global optimization method has been implemented. The local garbage sets always provide the processors with tasks and even with static load balancing processors always perform calculations. The algorithm has been tested on the various parallel systems.
5. The black box global optimization algorithm has been applied to solve three typical practical problems: multidimensional scaling, growth model of the human mandible and many-body.

Conclusions

The investigation of a simplex based Lipshitzian global optimization method may be summarized as follows:

- The versions of the method with the Lipshitzian bounds calculated using function values at vertices are more efficient than using function values at inner points since the same function values are attributed to neighboring and descendant simplices. It is not reasonable to calculate tight bounds taking into account all vertices since the cheaper bounds imply similar ratio of discarding of not prospective simplices.
- The partition of a simplex in two subsimplices defined by the middle point of the longest edge is more efficient than its partition in *n*+2 subsimplices using middle points of all edges. It is not reasonable to maintain regularity of the subsimplices while partitioning. The initial covering does not have much influence, although the vertex triangulation has been observed slightly better than the over-covering by regular simplices.
- The proposed irregular simplex based method performs better than the other known branch and bound algorithms with respect to the number of function evaluations.
- The master-slave paradigm is not efficient for parallel implementation of the proposed method since the calculation of bounds is cheap comparing with the time needed to send the task and the results. The communications are frequent, therefore master becomes a bottleneck and the slaves become idle when waiting for new tasks.
- Acceleration anomaly takes place for the parallel implementations with the distributed initialization. The reason is the dependence of distribution of simplices of initial partitioning on the number of processors.
- The proportion of time spent on calculations of the parallel implementations with the distributed initialization is better than that with the centralized initialization. This may be explained by the shortage of tasks in the latter case: only the root processor has tasks after the initialization while others are idle waiting for them. The search tree in case of central initialization is narrower than the search tree in case of distributed initialization.

- The proportion of time spent on calculations by the parallel implementation with static load balancing is better than by the implementation with dynamic load balancing since tasks are small with respect to the communication overheads. The communications cause an additional overhead, but the search tree is narrow and there are not enough tasks to share.

The investigation on black box global optimization inspired by the interval methods may be summarized as follows:

- The proposed method is reliable; in all testing cases the global minimum has been found. The method is applicable much widely than the interval methods therefore it is a good candidate to substitute the interval methods in a "black box" situation when interval methods can not be applied.
- The proposed method has been applied to several practical problems. It favorably competes with the methods originally used to solve these problems with respect to the criterion of calls of the objective function.
- The implementation of parallel version of the proposed black box global optimization method is efficient. Since the solution may be lost the algorithm is embedded into a loop which restarts the calculation. A local *outer* candidate sets always provide the processors with tasks, and processors always perform calculations even in case of static load balancing.

References

[1] **Ali M.M., Storey C. and Törn A.** Application of Stochastic Global Optimization Algorithms to Practical Problems. *Journal of Optimization Theory and Applications. Vol. 95, No. 3, 545-563*. 1997.

[2] **Andersen P., Nielsen M. and Kreiborg S.** 4D Shape – preserving modeling of bone growth. *Lecture Notes in Computer Science, Vol 1496, pp 710-719*. 1998.

[3] **Berner S.** Parallel Methods for Verified Global Optimization Practice and Theory. *Journal of Global Optimization 9, Kluwer Academic Publishers, 1-22*. 1996.

[4] **Bertsekas D.P. and Tsitsiklis J.N.** Parallel and Distributed Computation Numerical Methods. *Prentice Hall*. 1989.

[5] **Brenner D.W.** Empirical potential for hydrocarbons for use in simulating the chemical vapor deposition of diamond filters. *Physical Review B, Vol. 42, No. 15, 9458-9471*. 1990.

[6] **Borg I. and Lingoes J.** Multidimensional similarity structure analysis. *Springer, New York*. 1987.

[7] **Caprani O., Godthåb B. and Madsen K.** Use of a Real-Valued Local Minimum in Parallel Global Optimization. *Interval Computations 2, 71-82*. 1993.

[8] **Clausen J.** Parallel Search-Based Methods in Optimization. *In: Applied Parallel Computing – Industrial Computation and Optimization, Proceedings of PARA96, Springer Lecture Notes in Computer Science 1184, 176-185*. 1996.

[9] **Clausen J.** Parallel branch-and-bound – principles and personal experiences. *In: A. Migdalas, P.M. Pardalos and S. Storøy (eds), Parallel Computing in Optimization, Kluwer Academic Publishers, 239-267*. 1997.

[10] **Clausen J. and Perregaard M.** On the best search strategy in parallel branch-and-bound: Best-First Search versus Lazy Depth-First Search. *Annals of Operations Research 90, 1-17*. 1999.

[11] **Clausen J. and Träff J.L.** Implementation of parallel Branch-and-Bound algorithms – experiences with the graph partitioning problem. *Annals of Operations Research 33, 331-349*. 1991.

[12] **Clausen J. and Žilinskas A.** Global Optimization by Means of Branch and Bound with Simplex Based Covering. *Technical report. IMM-REP-1998-23, DTU*. 1998.

[13] **Čiegis R.** Lygiagretieji algoritmai. Vadovėlis aukštųjų mokyklų studentams, studijuojantiems matematiką ir informatiką. *V.: Technika*. 2002.

[14] **Čiegis R. and Šablinskas R.** Hyper-Rectangle Selection and Distribution Algorithm for Parallel Adaptive Numerical Integration. *Informatica, Vol.10, No.2, 161-170*. 1999.

[15] **Čiegis R., Šablinskas R. and Wasniewski J.** Numerical Integration on Distributed-Memory Parallel Systems. *Informatica, Vol.9, No.2, 123-140*. 1998.

[16] **Davison M.** Multidimensional scaling. *John Wiley, New York*. 1983.

[17] **DeLeeuw J.** Applications of Convex Analysis to Multidimensional Scaling. *In: J.R. Barra et al (eds). Recent Developments in Statistics, North Holland, 133-145*. 1977.

[18] **DeLeeuw J.** Convergence of the Majorization Method for Multidimensional Scaling. *Journal of Classification, 5, 163-180*. 1988.

[19] **Dzemyda G. and Petkus T.** Application of Computer Network to Solve the Complex Applied Multiple Criteria Optimization Problems. *Informatica, Vol.12, No.1, 45-60*. 2001.

[20] **Ferreira A. and Morvan M**. Models for Parallel Algorithm Design: an Introduction. *In: A. Migdalas, P.M. Pardalos and S. Storøy (eds), Parallel Computing in Optimization, Kluwer Academic Publishers*. 1997.

[21] **Floudas C. and Pardalos P.** Handbook of Test Problems in Local and Global Optimization. *Kluwer Academic Publishers*. 1999.

[22] **Gendron B. and Crainic T.G.** Parallel branch-and-bound algorithms: survey and synthesis. *Operations Research, Vol. 42, No. 6, 1042-1066*. 1994.

[23] **Gendron B. and Crainic T.G.** A parallel branch-and-bound algorithm for multicommodity location with balancing requirements. *Computers Ops Res, Vol. 24, No. 9, pp. 829-847*. 1997.

[24] **Gergel V.P.** Numerical Experiments with an Interval Method for Global Optimization on a Transputer System. *Report NI 90-08, Department of Mathematical Modeling, Technical University of Denmark, Lyngby, Denmark.* 1990.

[25] **Green P., Carmone F. and Smith S.** Multidimensional Scaling: Concepts and Applications. *Allyn and Bacon, Boston*. 1989.

[26] **Groenen P.** The majorization approach to multidimensional scaling, *DSWO Press*. 1993.

[27] **Grotzinger S.J.** Supports and Concave Envelopes. *Mathematical Programming, 31, 339-347*. 1985.

[28] **Hansen E.** A global convergent interval analytic method for computing and bounding real roots. *BIT 18, 415-424*. 1978.

[29] **Hansen E.** Global optimization using interval analysis – the multidimensional case. *Numerische Mathematik 34, 247-270*. 1978.

[30] **Hansen E.** Global Optimization using Interval Analysis. *Marcel Dekker, New York*. 1992.

[31] **Hansen P. and Jaumard B.** Lipshitz optimization, *In: R.Horst and P.Pardalos (eds.), "Handbook of Global Optimization", Kluwer Academic Publishers, Dodrecht, 404-493*. 1995.

[32] **Henriksen T. and Madsen K.** Use of Depth-First Strategy in Parallel Global Optimization. *Report NI-92-10, Department of Mathematical Modeling, Technical University of Denmark, Lyngby, Denmark*. 1992.

[33] **Hentenryck van P.** Numerica: a Modeling Language for Global Optimization. *MIT Press, Cambridge, Massachusetts*. 1997.

[34] **Hooker J.N.** Testing heuristics: we have it all wrong. *Journal of Heuristics 1, 33-42*. 1995.

[35] **Horst R., Pardalos P.M. and Thoai N.V.** Introduction to Global Optimization. *Kluwer Academic Publishers*. 1995.

[36] **Horst R. and Tuy H.** Global optimization: deterministic approaches. *Springer*. 1990.

[37] **INMOS Limited.** Occam 2 Reference Manual. *Prentice Hall*. 1988.

[38] **Jansson C.** Construction of convex lower and concave upper bound functions. *Technical Report 98.1, TU Hamburg-Harburg*. 1998.

[39] **Jansson C.** Convex-concave extensions. *BIT Numerical Mathematics Volume 40, Issue 2, 291-313*. 2000.

[40] **Jansson C. and Knüppel O.** A global minimization method: The multidimensional case. *Report 92.1, TU Hamburg-Harburg*. 1992.

[41] **Jansson C. and Knüppel O.** A Branch and Bound Algorithm for Bound Constrained Optimization Problems without Derivatives. *Journal of Global Optimization 7, Kluwer Academic Publishers, 297-331*. 1995.

[42] **Klock H. and Buhmann J.M.** Data visualization by multidimensional scaling: a deterministic annealing approach. *Pattern Recognition 33, 651-669*. 2000.

[43] **Knüppel O.** A PROFIL/BIAS implementation of a global minimization algorithm. *Report 95.4, TU Hamburg-Harburg*. 1995.

[44] **Lee E.K. and Mitchell J.E**. Computational Experience of an Interior-Point SQP Algorithm in a Parallel Branch-and-Bound Framework. *Frenk H. et al. (eds), High performance optimization. Kluwer Academic Publishers, 329-347*. 2000.

[45] **Madsen K.** Real versus Interval Methods for Global Optimization. *Presentation at the Conference "Celebrating the 60th Birthday of M.J.D. Powell", Cambridge*. July 1996.

[46] **Madsen K. and Zertchaninov S.** Branch-and-Bound for Global Optimization. *IMM-REP-1998-05, Department of Mathematical Modeling, Technical University of Denmark, Lyngby, Denmark*. 1998.

[47] **Madsen K. and Žilinskas J.** Test functions for global optimization. *http://www.imm.dtu.dk/~km/GlobOpt/testex/*. 1999.

[48] **Madsen K. and Žilinskas J.** Testing of attraction based subdivision and interval methods for global optimization. *IMM-REP-2000-04, Department of Mathematical Modeling, Technical University of Denmark, Lyngby, Denmark*. 2000.

[49] **Madsen K. and Žilinskas J.** Evaluating Performance of Attraction Based Subdivision Methods for Global Optimization. *Second International Conference "Simulation, Gaming, Training, Business Process Reeingineering in Operations", RTU, Riga, 38-42*. 2000.

[50] **Mans B., Mautor T. and Roucairol C.** A parallel depth first search branch and bound algorithm for the quadratic assignment problem. *European Journal of Operational Research 81, 617-628*. 1995.

[51] **Mans B. and Roucairol C.** Performance of parallel branch and bound algorithms with best-first search. *Discrete Applied Mathematics 66, 57-74*. 1996.

[52] **Mathar R.** A Hybrid Global Optimization Algorithm for Multidimensional Scaling. *Classification and Knowledge Optimization. Proceedings of the 20th Annual Conference of the Gesellschaft für Klassification e.V., University of Freiburg, 63-71*. 1996.

[53] **Mathar R. and Žilinskas A.** On global optimization in two – dimensional scaling. *Acta Appl. Math, v.33, N.1, 109-118*. 1994.

[54] **McKeown G.P., Rayward-Smith V.J., Rush S.A. and Turpin H.J.** Using a Transputer network to solve Branch-and-Bound problems. *Transputing 91, Proceedings of the World Transputer User Group Conference, 781-800*. 1991.

[55] **Message Passing Interface Forum**. MPI: A Message-Passing Interface standard (version 1.1). *Technical report*, 1995.

[56] **Migdalas A., Pardalos P. and Storrey S., eds.** Parallel computing in optimization. *Kluwer*. 1997.

[57] **Mockus J.** Bayesian approach to global optimization. *Kluwer Academic Publishers*. 1989.

[58] **Moore R.E.** Interval Analysis. *Prentice-Hall*. 1966.

[59] **Moore R.E.** A test for existence of solutions to non-linear systems. *SIAM J. Num. Anal. 14, 611-615*. 1977.

[60] **Moore R.E.** Methods and Applications of Interval Analysis. *SIAM, Philadelphia*. 1979.

[61] **Pardalos P., Horst R. and Thoi N.** Introduction to Global Optimization. *Kluwer Academic Publishers*. 1995.

[62] **Perttunen C., Jones D. and Stuccman B.** Lipshitz optimization without the Lipshitz constant. *Journal of Optimization Theory and Applications, 79, 157-181*. 1993.

[63] **Petkus T.** Kompiuterių tinklo panaudojimas interaktyviame optimizavime. *Daktaro disertacija, VPU*. 2001.

[64] **Powell M.J.D.** A Hybrid Method for Non-Linear Equations. *in "Numerical Methods for Nonlinear Algebraic Equations", P. Rabinowitz, ed. 87-114*. 1970.

[65] **Rayward-Smith V.J., Rush S.A. and McKeown G.P.** Efficiency considerations in the implementation of parallel branch-and-bound. *Annals of Operations Research 43, 123-145*. 1993

[66] **Rinnooy Kan A.H.G. and Timmer G.T.** Global Optimization. *In "Handbook in Operations Research Vol 1: Optimization", Nemhauser et.al., eds. 631-659.* 1989.

[67] **Schittkowski K.** More Test Examples for Nonlinear Programming Codes. *Lecture Notes in Economics and Mathematical Systems. Springer, New York.* 1987.

[68] **Schwefel H.P.** Evolution and optimal seeking. *John Wiley*. 1995.

[69] **Skelboe S.** Computation of Rational Interval Functions. *BIT 14, 87-95*. 1974.

[70] **Sørevik T.** A Programmer's View of Parallel Computers. *In: A. Migdalas, P.M. Pardalos and S. Storøy (eds), Parallel Computing in Optimization, Kluwer Academic Publishers*. 1997.

[71] **Stein R. M.** T800 and Counting. *BYTE, 287-296, November* 1988.

[72] **Šablinskas R.** Lygiagrečiųjų algoritmų tyrimas paskirstytos atminties lygiagretiems kompiuteriams. *Daktaro disertacija, VDU*. 1999.

[73] **Šaltenis V.** Structure Analysis of Optimization Problems. *Mokslas, Vilnius*. 1989. (in Russian).

[74] **Tersoff J.** New empirical approach for the structure and energy of covalent systems. *Physical Review B, Vol. 37, No. 12, 6991-7000*. 1988.

[75] **Tersoff J.** Empirical interatomic potential for silicon with improved elastic properties. *Physical Review B, Vol. 38, No. 14, 9902-9905*. 1988.

[76] **Tingleff O.** Getting to Know the Transputer System. *Report 90-05, Department of Mathematical Modeling, Technical University of Denmark, Lyngby, Denmark*. 1990.

[77] **Tingleff O., Žilinskas A. and Žilinskas J.** A Two Dimensional Optimization Algorithm Based on New Statistical Model of Multimodal Functions. *In: C. Carlsson et al. (eds.), Global & Multiple Criteria Optimization and Information System Quality, Åbo Akademi Press, Finland, 103-112*. 1998.

[78] TOP500 Supercomputer Sites. *http://www.top500.org*/.

[79] **Törn A., Ali M. and Viitanen S.** Global optimization: problem classes and solution techniques. *JOGO, v.4, N.14, 437-447*. 1999.

[80] **Törn A. and Žilinskas A.** Global Optimization. *Lecture Notes in Computer Science, v. 355, Springer*. 1989.

[81] **Willebeek-LeMair M. and Reeves A.P.** Strategies for Dynamic Load Balancing on Highly Parallel Computers. *IEEE Transactions on Parallel and Distributed Systems, Vol. 4, No. 9*. 1993.

[82] **Wilkinson B. and Allen M.** Parallel programming: techniques and applications using networked workstations and parallel computers. *Prentice-Hall*. 1999.

[83] **Wilson P.** Parallel Processing Comes to PCs. *BYTE, 213-218, November* 1988.

[84] **Wood G.R. and Zhang B.P.** Estimation of the Lipschitz Constant of a Function. *Journal of Global Optimization. Kluwer Academic Publishers. 8: 91-103*. 1996.

[85] **Yao X. and Liu Y.** Fast evolution strategies. *Control and Cybernetics 26, 467-496*. 1997.

[86] **Žilinskas J.** Optimization of Lipschitzian functions by simplex based branch and bound. *Information Technology and Control No.1(14), Kaunas, Technologija, 45-50*. 2000.

[87] **Žilinskas J.** Black Box Global Optimization Inspired by Interval Methods. *Information Technology and Control No.4(21), Kaunas, Technologija, 53-60*. 2001.

[88] **Žilinskas J.** Programs for global optimization. *http://www.dsplab.ktu.lt/~jzil/*.