Parallel global optimization with deterministic approaches

Wu, Yong

2006

Wu, Y. (2006). Parallel global optimization with deterministic approaches. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/5264>

<https://doi.org/10.32657/10356/5264>

Nanyang Technological University

Downloaded on 26 Apr 2023 16:54:10 SGT

PARALLEL GLOBAL OPTIMIZATION WITH DETERMINISTIC APPROACHES

WU YONG

School of Mechanical and Aerospace Engineering

A thesis submitted to the Nanyang Technological University
in fulfillment of the requirement for the degree of
Doctor of Philosophy

2006

Abstract

Global optimization is important both in theory and practical applications. The objectives of the research conducted in this thesis are two-fold. The first one is to construct methods which aim to reduce the effort needed to solve global optimization problems. The second objective is to investigate suitable parallel models for those methods to speed up the computation, which is especially important for time-consuming problems. Two partitioning approaches are proposed in this research for unconstrained and constrained problems, respectively.

For unconstrained optimization, a new global optimization approach, TRIOPT (TRIangulated OPTimization), is proposed. TRIOPT is based on partitioning the feasible region $\mathcal{D}$ geometrically by triangulation so that search can be conducted in smaller subregions of interest. This consequently increases the reliability in locating the global optimum and reduces the number of function evaluations required. The partitions (simplices) are assessed by the aggregated entropies calculated from the transformed values of the vertices, which lead to a dynamic degree of parallel search in the feasible region $\mathcal{D}$. A novelty in the search scheme is that once a partition narrows down to a small size, its vertices are eliminated from the available sample set. This change of global information triggers a re-calculation of transformed values and guides the search to new subregions. TRIOPT employs a Master/Slave architecture for parallel processing according to its algorithmic characteristics. The master processor controls the dynamic degree of parallelization, and the slave processors evaluate the samples.

In constrained optimization, interval analysis is applied since it can conveniently check the feasibility of a certain search area rather than a single

point. Search schemes are constructed based on branch and bound concepts. Three interval subdivision strategies are introduced for COPIA (Constrained OPTimization using Interval Analysis). Those strategies only utilize the information of the objective and constraints functions and do not require the calculation of gradients. Such features make them advantageous when the gradient information is either not available or very expensive to evaluate. Furthermore, one of the three strategies bases only on the information of the objective function, which enables it applicable to unconstrained optimization. The parallel models for COPIA are decentralized parallel computation models applied with blocking message passing and non-blocking message passing, respectively. The advantage of the models is that there is no need to design a special initial partition, no need to consider the problem of load balancing, and the management of communication is simple. Numerical results show that the two parallel models can significantly speed up the computation. Since there is no special assumption about the problems under consideration, the models can also be applied to general branch and bound problems.

Acknowledgements

I would like to express my sincerest gratitude and appreciation to my supervisor, Associate Professor Arun Kumar, for his invaluable advices and continued direction. I am particularly grateful to Dr. Linet Özdamar, who co-supervised me for about two and a half years, for her constant help and all the fruitful and inspiring discussions between us. Their patience, understanding and encouragement have enabled me to pursue my research effectively.

I would like to thank the Division of Systems and Engineering Management, School of Mechanical and Aerospace Engineering, and Nanyang Technological University, for their consistent support, both educationally and financially, for my research.

Thanks to Singapore National Grid Pilot Platform (NGPP) for providing the computation resources for my research. Miss Beibei Jiang from National Grid Support Unit and Singapore Computer System Limited helped me set up the computation environment and provided lots of advice during my parallel code developing phase, great thanks to her.

Special thanks to the professors, staff, and colleagues in the Centre for Supply Chain Management (CSCM), for their assistance and kind cooperation. It is a great pleasure to study and work with them in such a pleasant research environment.

Comments and suggestions from thesis reviewers helped a lot in the modification of my thesis that led to significant improvement. I am grateful for their time and effort spent during the review process and their pertinent and inspiring advice for the revision of the thesis.

Finally, I would like to thank my wife, my parents, my family, and all my friends who have been extremely supportive during my study and research,

especially during the hard times I went through. My gratitude to them far exceeds these few words can express; without their support, this work would not have been possible.

Table of Contents

Abstract	i
Acknowledgements	iii
Table of Contents	v
List of Figures	ix
List of Tables	xii
List of Algorithms	xiii
Notation	xiv
1 Introduction	1
1.1 Problem Formulation	1
1.2 Partitioning Approaches	3
1.3 Parallel Computation	4
1.4 Objectives and Approaches	6
1.5 Organization	7
2 Literature Review	9
2.1 Direct Search Methods	9

2.2	Lipschitz Optimization	11
2.3	Adaptive Partitioning Methods	14
2.4	Response Surface Methods	15
2.5	Interval Methods	17
2.6	Other Methods	20
2.7	Summary	21
3	Triangulated Optimization	23
3.1	Motivation	23
3.2	Simplex Assessment	26
3.2.1	Function Transformation	26
3.2.2	Simplex Entropy	27
3.3	Partitioning Scheme	28
3.3.1	Initial Partition	28
3.3.2	Partitioning Strategies	29
3.4	The Algorithm	31
3.4.1	Algorithm Description	31
3.4.2	Complexity Analysis	33
3.4.3	Convergence Analysis	34
3.5	Numerical Experiments	36
3.5.1	Test Problems	36
3.5.2	Methods for Comparison	39
3.5.3	Implementation Details	41
3.5.4	Parameter Sensitivity Analysis	43
3.5.5	Numerical Results	49
3.6	Parallel Implementation	57
3.6.1	Brief Introduction	57

3.6.2	The Parallel Model	60
3.6.3	Parallel Performance	64
3.7	Summary	68
4	Constrained Optimization Using Interval Analysis	69
4.1	Introduction	69
4.2	New Interval Subdivision Strategies	73
4.2.1	Subdivision Based on Information Gain	74
4.2.2	Subdivision Based on Constraint Violation	80
4.2.3	Subdivision Based on Infeasibility Index	82
4.3	The Algorithm	83
4.4	Numerical Experiments	84
4.4.1	Test Problems	84
4.4.2	Implementation Details	84
4.4.3	Numerical Results	86
4.5	Parallel Implementation	90
4.5.1	Parallel Computation Model Topology	91
4.5.2	Blocking Message Passing Model	91
4.5.3	Blocking Message Passing Performance	98
4.5.4	Non-Blocking Message Passing Model	109
4.5.5	Non-Blocking Message Passing Performance	114
4.6	Summary	122
5	Conclusions and Future Recommendations	124
5.1	Conclusions	124
5.2	Recommendations for Future Research	127
	References	130

Appendix	144
A Test Problems for TRIOPT	144
A.1 Theoretical Problems	144
A.2 Engineering Applications	150
B Test Problems for Constrained Optimization	156
B.1 First Batch of Problems	156
B.2 Second Batch of Problems	162

List of Figures

3.1	An illustration of splitting efficiency in 3D.	24
3.2	A GIS Triangulated Irregular Network (TIN).	25
3.3	Delaunay <i>vs.</i> non-Delaunay triangulation.	29
3.4	Performance comparison with 1.0% accuracy level	54
3.5	Performance comparison with 0.01% accuracy level	55
3.6	The parallel computation model for TRIOPT.	61
3.7	Parallel evaluations on Six-Hump with 1.0% accuracy	65
3.8	Parallel evaluations on Six-Hump with 0.01% accuracy	65
3.9	Speedup on the Six-Hump problem.	66
3.10	Speedup on the Rastrigin(5) problem.	66
4.1	Split from the widest edge.	75
4.2	Split by information gain.	76
4.3	Ratio of function evaluations	88
4.4	Ratio of constraint evaluations	89
4.5	Ratio of maximum list length	89
4.6	The ring topology of the parallel model.	91
4.7	A typical parallel computation cycle	92
4.8	Ratios for Problem 1	101
4.9	Ratios for Problem 2	101

4.10 Ratios for Problem 3	102
4.11 Ratios for Problem 4	102
4.12 Ratios for Problem 5	103
4.13 Ratios for Problem 6	103
4.14 Ratios for Problem 7	104
4.15 Ratios for Problem 8	104
4.16 Ratios for Problem 9	105
4.17 Ratios for Problem 10	105
4.18 Speedup for Problems 1, 7 and 8 (blocking)	107
4.19 Speedup for Problems 3, 6 and 10 (blocking)	107
4.20 Speedup for Problems 4 and 5 (blocking)	108
4.21 Speedup for Problems 2 and 9 (blocking)	108
4.22 Speedup for Problems 1, 2 and 8 (non-blocking)	118
4.23 Speedup for Problems 3 and 6 (non-blocking)	118
4.24 Speedup for Problems 4 and 5 (non-blocking)	119
4.25 Speedup for Problems 7 and 10 (non-blocking)	119
4.26 Speedup for Problems 9, 11 and 13 (non-blocking)	120
4.27 Speedup for Problems 12 and 14 (non-blocking)	120
4.28 Speedup for Problem 15 (non-blocking)	121
4.29 Speedup for Problem 16 (non-blocking)	121
 A.1 Surface plot of Six-Hump.	 145
A.2 Surface plot of G & P.	145
A.3 Surface plot of Ackley.	146
A.4 Surface plot of Spike.	147
A.5 Surface plot of Schwefel.	147
A.6 Surface plot of Easom.	148

A.7 Surface plot of Crescent.	148
A.8 Surface plot of Griewank(2).	149
A.9 Surface plot of Rastrigin(2).	150
A.10 Optimal shape design of a truss.	151
A.11 Cylindrical pressure vessel.	153

List of Tables

3.1	Summary of test problems for TRIOPT.	38
3.2	Performance of different combinations of α and β	44
3.3	Different combinations of p and q at 1.0% accuracy level . . .	46
3.4	Different combinations of p and q at 0.01% accuracy level . . .	47
3.5	Percentage of different combinations of p and q	48
3.6	Results for test problems with 1.0% accuracy level.	51
3.7	Results for test problems with 0.01% accuracy level.	52
3.8	Results for Protein folding problems	56
3.9	Number of parallel function evaluations.	67
3.10	Parallel evaluations for Protein folding problems	68
4.1	Performance of information gain on unconstrained problems. .	79
4.2	Summary of constrained test problems.	85
4.3	Test results of first batch problems	87
4.4	Performance with blocking message passing	99
4.5	Time used with blocking message passing	106
4.6	Test results using non-blocking message passing	116
4.7	Time used with non-blocking message passing	117

List of Algorithms

3.1	TRIOPT	32
3.2	Parallel TRIOPT—Master	62
3.3	Parallel TRIOPT—Slave	63
4.1	Interval analysis for constrained optimization	70
4.2	Direction selection based on information gain	78
4.3	Direction selection based on constraint violation	81
4.4	P-COPIA using blocking message passing	94
4.5	P-COPIA using non-blocking message passing	111
4.6	Recv_Message_Processing ()	113
4.7	Box_Processing ()	113
4.8	Send_Message_Processing ()	113

Notation

Throughout the Thesis

$\underline{\mathbf{B}}$	Lower bounds of all variables.
$\overline{\mathbf{B}}$	Upper bounds of all variables.
$c(\cdot)$	General constraints.
$\mathcal{D}$	Search region.
$E(p)$	Efficiency using p processors.
$f(\cdot)$	Objective function.
$g(\cdot)$	Inequality constraint.
$h(\cdot)$	Equality constraint.
L	Lipschitz constant.
n	Number of variables.
P	Processor.
p	Number of parallel processors.
$\mathbb{R}$	Set of reals.
r	Number of inequality constraints.
$S(p)$	Speedup using p processors in parallel.
s	Number of total constraints.
T_{serial}	Solution time using a single processor.
T_{parallel}	Solution time using multiple processors.

x	A scalar value of coordinate.
x_i	The i^{th} component of vector $\mathbf{x}$.
$\mathbf{x}$	Vector of coordinates.
$\mathbf{x}^*$	Location of a global maximum.
$\mathbf{x}'$	Location of a local maximum.
$\mathbf{x}^\varepsilon$	Location of an ε -maximum.

For Unconstrained Optimization

α	Cut value for transformation.
β	Cut value for simplex selection.
δ	Area tolerance level.
E	Entropy for a simplex.
$\dot{f}(\cdot)$	Linear transformation function.
$\ddot{f}(\cdot)$	Nonlinear transformation function.
$f_{\max}$	Current maximum function value in set $\mathcal{V}$.
$f_{\min}$	Current minimum function value in set $\mathcal{V}$.
$H(\cdot)$	Hyper-volume.
L_u	Shortest edge length of a simplex .
$\mathcal{L}_{nm}$	Local search list.
$\mathcal{L}_p$	Potential list.
$\mathcal{L}_t$	Temporary list.
$\mathcal{L}_w$	Working list.
P_s	Splitting performance.
p	Inflation coefficient in transformation function.
q	Deflation coefficient in transformation function.
r	Sample bias regulator.

$\mathcal{S}$	Simplex.
t	Threshold in transformation function.
$\mathcal{V}$	Set of samples (points).
$\mathbf{x}_{\text{new}}$	Partitioning point in a simplex.

For Constrained Optimization

$\mathcal{B}$	Box (interval).
$C(\cdot)$	Inclusion function of general constraints $c(\cdot)$.
d	Split direction.
ε	Constraint tolerance.
ε_f	Objective function tolerance.
ε_w	Edge width tolerance.
$F(\cdot)$	Inclusion function of objective function $f(\cdot)$.
$G(\cdot)$	Inclusion function of inequality constraints $g(\cdot)$.
$H(\cdot)$	Inclusion function of equality constraint $h(\cdot)$.
I_F	Infeasibility index.
I_G	Information gain.
$\mathcal{L}$	Working list.
R	Constraint-variable relationship matrix.
V	Constraint violation.
$w(\cdot)$	Width function.
X	Interval over $\mathbf{x}$.

Chapter 1

Introduction

1.1 Problem Formulation

Optimization, which can be roughly divided into two categories: local optimization and global optimization, is used in a variety of areas. For local optimization lots of efficient methods already exist. Global optimization is more complicated, it is not enough to only rely on local information—the best solution of all local optima should be determined. In case of a complex nonlinear problem, the number of local optima is often large and is unknown beforehand, which makes distinguishing the global optimum from local optima a difficult task to accomplish. At the same time, the fact that the quality of local and global solutions may differ substantially indicates utilization of global optimization is not only necessary but also essential.

Optimization problems are often described through mathematical models which normally consist of a group of equations. Some models can provide both function value and gradient information; while others can only provide function value information, the derivative is either not available or very ex-

1.1 Problem Formulation

pensive to compute. Sometimes models are even not given explicitly, that is, the information is given computationally—feed in an input, generate an output. Those models are described as “black box” problems.

The global optimization problem expressed below is considered in this research:

$$\begin{aligned}
 & \max f(\mathbf{x}) \\
 & \text{s.t. } g_i(\mathbf{x}) \leq 0, \quad i = 1, \dots, r \\
 & \quad h_j(\mathbf{x}) = 0, \quad j = r + 1, \dots, s \\
 & \quad \mathbf{x} \in \mathcal{D}.
 \end{aligned} \tag{1.1}$$

where $\mathcal{D} \subset \mathbb{R}^n$ is the search domain, normally is confined by the lower bounds $\underline{\mathbf{B}}$ and the upper bounds $\overline{\mathbf{B}}$ of all variables, that is, $\mathcal{D} \subseteq [\underline{\mathbf{B}}, \overline{\mathbf{B}}]$, which is also called an n -dimensional hypercube. $f : \mathcal{D} \rightarrow \mathbb{R}$ is the objective function. $g_i : \mathcal{D} \rightarrow \mathbb{R}$, $i = 1, \dots, r$ and $h_j : \mathcal{D} \rightarrow \mathbb{R}$, $j = r + 1, \dots, s$ are the inequality and equality constraints; they are denoted as c_i , $i = 1, \dots, s$ when there is no need to differentiate the type of constraints. In this research, Equation (1.1) is called a Constrained Optimization Problem (COP) when constraints c_i are presented; otherwise, it is called an Unconstrained Optimization Problem (UOP), although the bounds $[\underline{\mathbf{B}}, \overline{\mathbf{B}}]$ are still effective. For both COP and UOP, the information of derivative is assumed to be not available and the evaluations of objective function and constraints are costly and time-consuming.

The concepts of *global maximum*, *local maximum* and ε -*maximum* are defined as follows:

Definition 1.1. A **global maximum** for Equation (1.1) is defined as a point $\mathbf{x}^*$ such that $f(\mathbf{x}^*) \geq f(\mathbf{x})$ for all $\mathbf{x} \in \mathcal{D}$.

Definition 1.2. A **local maximum** for Equation (1.1) is defined as a point $\mathbf{x}'$ such that $f(\mathbf{x}') \geq f(\mathbf{x})$ for all $\mathbf{x} \in \mathcal{D} \cap N_\epsilon(\mathbf{x}')$. Here $\epsilon > 0$ and $N_\epsilon(\mathbf{x}')$ is an ϵ -neighborhood around $\mathbf{x}'$ defined by $N_\epsilon(\mathbf{x}') = \{\mathbf{x} \mid \|\mathbf{x} - \mathbf{x}'\| < \epsilon\}$.

Definition 1.3. An ϵ -**maximum** for Equation (1.1) is defined as a point $\mathbf{x}^\epsilon \in \mathcal{D}$ such that $f(\mathbf{x}^\epsilon) \geq f(\mathbf{x}^*) - \epsilon$. Here $f(\mathbf{x}^*)$ is a global maximum, and $\epsilon > 0$.

Generally, f , g and h are nonlinear and non-convex functions and the feasible set is non-convex. Theoretically, all global maxima for Equation (1.1) are wanted, tacitly assuming that the solution set is countable. However, this goal might not be achievable due to computational reasons; a more practical aim of global optimization is to find suitable ϵ -maxima with a sufficient small ϵ instead of finding all global maxima.

1.2 Partitioning Approaches

A large class of global optimization algorithms have adopted Divide and Conquer (Partitioning) approaches. Partitioning approaches divide given domains into smaller sub-spaces whose potential of holding the global optimum is determined either reliably (deterministic approaches), such as Hansen and Jaumard (1995) and Pintér (1996); or in a heuristic manner (stochastic approaches), such as Özdamar and Demirhan (2000, 2001).

Deterministic partitioning algorithms can be classified under two major categories. The first category includes algorithms based on interval methods, such as Hansen (1992), Kearfott (1996a), and Moore and Ratschek (1988). These algorithms use inclusion functions where the range of f within a specific partition is estimated by an inclusion function that always covers actual

function range, hence, leading to reliable box disposal. The second category includes algorithms based on certain *a priori* assumptions on functions, such as Lipschitz methods, e.g., (Gourdin et al., 1994; Hansen and Jaumard, 1995; Pintér, 1988), which generate a partition of the domain based on an assumed rate of change, that is, the Lipschitz constant. If the constant is known with accuracy, then these methods are also reliable.

Stochastic partitioning methods assess each partition with certain criteria and subdivide those partitions which are more “promising”. Sub-spaces can be selected based on possible largest expected increase in function values (Tang, 1994); or based on fuzzy assessment (Demirhan and Özdamar, 1999; Özdamar and Demirhan, 2000) where evidence within each partition is collected by using random search techniques; among others.

Normally partitioning approaches are carried out by rectangular splitting scheme. However, there are also simplicial partitioning algorithms that are at times more efficient in terms of the number of new partitions obtained per additional sample taken. As indicated in Clausen and Žilinskas (2002), Wood (1991), and Zhang et al. (1993), simplicial partitioning might be comparatively more flexible because a wide range of partitioning schemes can be used to result in more regular geometrical shapes that are proportional to the topology of the domain. Similar to rectangular partitioning, simplicial partitioning is also convergent when nested partitioning is applied, because edge length is reduced to zero exponentially.

1.3 Parallel Computation

Despite the advanced computer support we have at hand, real world optimization problems are still challenging and computationally expensive to

1.3 Parallel Computation

solve. More efficient algorithms are required to reduce the effort to solve those problems. On the other hand, more powerful computation capacities are desired to speedup the processes of finding global optimal solutions.

Parallel computation, whose main purpose is to perform computations faster than they can be done with a single processor by using a number of processors concurrently, has had a tremendous influence on almost all the activities related to computing. The need for faster solutions and for solving larger-size problems arises in a wide variety of applications. Global optimization can be one of such beneficiaries of parallel computation, especially when the evaluations of the objective and constraint functions are time-consuming. At the same time, as problem size increases, the complexity of managing data is also expanding; sometimes such complexities increase exponentially. Parallel computation can lower the complexity since one processor might only need to manage a fraction of the whole data set.

Several alternative parallel architectures were developed throughout the 1980s and 1990s. A classification of computer architectures was proposed by Flynn (1972). Four classes were proposed based on the interaction between instruction and data streams of the processors:

- SISD: Single Instruction stream Single Data stream.
- SIMD: Single Instruction stream Multiple Data stream.
- MISD: Multiple Instruction stream Single Data stream.
- MIMD: Multiple Instruction stream Multiple Data stream.

Algorithms can select suitable architecture based on their algorithmic characteristics when parallelizing.

1.4 Objectives and Approaches

The research objectives of this thesis are twofold. The first one is to construct methods which aim to reduce the effort needed to solve global optimization problems. The second objective is to investigate suitable parallel models for those methods to speedup the computation, which is especially important for time-consuming problems. Two approaches, one for unconstrained optimization and the other for constrained optimization, together with their parallel computation models, are proposed in this research.

For unconstrained optimization, a dynamic parallel simplicial partitioning approach, which is called TRIOPT (TRIangulated OPTimization), is introduced. TRIOPT neither utilizes the information of derivative, nor makes any assumptions of the objective function, which makes it capable of dealing with “black box” problems. The novelty in TRIOPT is that simplices are assessed for re-partitioning after vertex function values are transformed and mapped onto the $[0, 1]$ interval. Adaptive parameters are utilized during function transformation. For instance, such a parameter is the threshold whose value can increase or decrease during the search. The threshold acts as a virtual platform where function values over it are treated as peaks and those below as valleys. Hence, a changing threshold re-defines peaks and valleys leading to dynamic re-direction during the search and a changing degree of parallelism. The adaptive threshold scheme is enabled by the removal of small simplices and their vertices from the available sample set as well as the acquisition of new sample data. Unlike other partitioning approaches where the current upper bound always improves during the search, the removal of simplices sometimes leads to a decrease in the threshold value rather than an increase. One limitation of TRIOPT is that it needs all vertices to cover

the whole search space, which only allows it work for low dimensional global optimization problems. The algorithmic characteristics of TRIOPT make it suitable to adopt an MIMD parallel computation architecture.

For constrained optimization, sub-spaces can no longer be selected only according to their objective function values; the constraints must be taken into consideration. A partition should be split only when there is at least one possibly feasible solution inside it. The disadvantages of point-valued partitioning approaches are: (a) to prove the existence of a possibly feasible solution in a partition is difficult, and (b) the selected partitioning point may not be feasible, which of course should not be evaluated and consequently gives rise to difficulties in assessing the partition. However, interval analysis provides a way to check the possible feasibility of a partition which overcomes the difficulties created by the constraints using partitioning approaches. The box partitioning scheme, which corresponds to the definitions and operations of interval arithmetics, is adopted for constrained optimization. Several subdivision direction selection strategies are proposed and combined with a best-first box selection strategy to solve constrained problems. These approaches are abbreviated as COPIA (Constrained OPTimization using Interval Analysis). COPIA is basically a branch and bound approach which is suitable to adopt the SIMD parallel computation architecture.

1.5 Organization

The organization of the thesis is as follows: Chapter 2 briefly reviews previous work related to this research. The unconstrained optimization approach—TRIOPT—is described in Chapter 3, where the motivation, simplex partitioning schemes and simplex assessment are presented; the algorithm de-

scription of TRIOPT is illustrated. The corresponding serial and parallel computation models are explained and followed by numerical studies to demonstrate the performance of TRIOPT. Chapter 4 first introduces three different interval subdivision direction selection strategies. Then the three strategies are applied to solve constrained problems and their performance is illustrated with numerical experiments. Parallel models utilizing blocking and non-blocking message passing are subsequently introduced and their performance analyzed. Chapter 5 summarizes the work done in this research and suggests some possible extensions. In Appendices A and B, detailed descriptions of test problems for unconstrained optimization and constrained optimization are provided, respectively.

Chapter 2

Literature Review

In the past three decades methods that aim to find the global optimum of a function have attracted a lot of interest. Numerous algorithms have been studied and can be found in surveys, such as Törn and Žilinskas (1989), Horst and Pardalos (1995) and Neumaier (2004), or in the introductory textbook by Horst et al. (1995). Here, a brief review of methods related to this research is provided.

2.1 Direct Search Methods

The phrase “direct search” was coined in Hooke and Jeeves (1961). Direct search methods are “derivative free” methods which are characterized by a series of exploratory moves that consider the behavior of the objective function at a pattern of points. This makes them advantageous when only the function information is available. They are local search techniques that explore the domain in a prescribed manner, moving from one solution to a hopefully better one.

A well-known direct search method is the pattern search method that acts

according to a given design of exploratory moves that consider the behavior of the objective function at a pattern of points, all of which lie on a rational lattice (Hooke and Jeeves, 1961; Lewis and Torczon, 1999).

Simplicial direct search methods, which make their moves in the search domain according to a simplicial pattern, are quite popular and widely used by researchers and practitioners (Walters et al., 1991). One example is the Nelder-Mead method (Nelder and Mead, 1965), which conducts its search using four operations: *reflection*, *expansion*, *contraction* and *shrinking* based on the function value of each vertex of the simplex. The Nelder-Mead simplex algorithm has enjoyed enduring popularity of all the direct search methods; it is the one most often found in numerical software packages (Press et al., 1992; Lagarias et al., 1998).

There are also methods which try to accelerate the search by constructing directions designed to use information about the curvature of the objective function obtained during the course of search. The exploratory direction is updated after a certain number of function evaluations, normally in a stage-wise manner. Rosenbrock's method (Rosenbrock, 1960) consists of a number of exploratory searches along a set of directions that is fixed for a given stage, but which is updated from stage to stage to make use of information acquired during the search. Powell's method (Powell, 1964) includes a sequence of $n+1$ one-dimensional searches in a stage. The one-dimensional searches are conducted by finding the exact minimizer of a quadratic interpolate computed for each direction. The first n searches are along each of a set of linearly independent directions. The last search is along the direction connecting the point obtained at the end of the first n searches with the starting point of the stage. At the end of the stage, one of the first n search directions is replaced

with the last search direction.

Recently, some local convergence proofs of pattern search algorithms have been given in Torczon (1997) and Dolan et al. (2000), among others. Lagarias et al. (1998) provide a convergence proof for strictly convex functions of the Nelder-Mead method in low dimensions. Coope and Price (2001) provide a general convergence analysis of grid-based methods—the convergence of direct search methods in the case where the underlying methods can be interpreted as a grid or pattern search over successively refined meshes for unconstrained optimization. However, the Nelder-Mead simplex method does not belong to this category of grid-based methods and its instability in convergence should be noted (McKinnon, 1998). Though researchers have tried to compensate for Nelder-Mead’s deficiencies in convergence, such as, Price et al. (2002) and Tseng (1999), many users are reported to prefer using multi-start Nelder-Mead to avoid this problem.

Recent reviews of direct search methods can be referred to Wright (1996), Trosset (1997), Lewis et al. (2000) and Kolda et al. (2003) and so on. Although direct search methods are local optimization techniques, they can improve their performance if applied with properly designed multi-start or parallel procedures, such as the Parallel Direct Search (PDS) (Dennis Jr. and Torczon, 1991) and the Asynchronous Parallel Pattern Search (APPS) (Hough et al., 2002) and so on.

2.2 Lipschitz Optimization

Lipschitz optimization solves global optimization problems in which the objective function and constraints are given explicitly and have bounded slopes.

That is, for any two points $\mathbf{x}_1$ and $\mathbf{x}_2$ in the feasible region of the problem,

$$|f(\mathbf{x}_1) - f(\mathbf{x}_2)| \leq L\|\mathbf{x}_1 - \mathbf{x}_2\|,$$

is satisfied for the objective function f , where, L is the *Lipschitz constant*. Similar properties also hold for constraints.

Many algorithms have been investigated based on the Lipschitzian model. In Hansen and Jaumard (1995) a good overview of Lipschitz optimization methods is presented. However, most of them require knowing the Lipschitz constant *a priori*. This prevents the implementation of Lipschitz optimization methods in real applications to some extent, because normally the Lipschitz constant is not available when a problem is being solved, or it cannot be known with accuracy.

Recently, some methods which do *not* require *a priori* Lipschitz constant have been proposed. Jones et al. (1993) introduce an approach called DIRECT (DIviding RECTangles), which is a modification of the standard Lipschitzian approach but eliminates the need to specify the Lipschitz constant. Concurrent searches are carried out using all possible constants from zero to infinity. In particular, the Lipschitz constant is viewed as a weighting parameter that indicates how much emphasis to place on global versus local search. DIRECT conducts dynamic parallel partitioning on boxes that are non-dominated with respect to two criteria: box size (representing unexplored areas—a global search feature) and box value (representing function fitness—a local search feature). Once the global search finds the basin of convergence of the optimum, the local search automatically exploits it. Due to the adaptive concurrent partitions selection strategy of DIRECT, it can be easily implemented in parallel. Watson and Baker (2001) provide a fully dis-

tributed parallel computation model for DIRECT. In He et al. (2002), a set of dynamic data structures to balance the memory requirements with execution time and simultaneously adapting to arbitrary problem size is presented, which makes DIRECT applicable to large-scale problems.

Another powerful algorithm that integrates deterministic and probabilistic global and local search within a global partitioning framework is the Lipschitz(-Continuous) Global Optimizer (LGO), which is a commercial software package developed by Pintér (Pintér, 1996, 1997). LGO solves global optimization problems under mild-continuity or Lipschitz continuity assumptions. It integrates several global (adaptive partitioning and random search based) and local (conjugate directions type) strategies and can be applied to “black box” functions. A summary of key features and usage of the LGO modeling and solver system can be found in Pintér (2002).

From the definition of Lipschitz constant, it seems that assessment of boxes is more suitable for Lipschitz calculation and therefore suitable for partition operations. However, there are also simplicial partitioning algorithms that are at times more efficient in terms of the number of new partitions obtained per additional sample taken. As indicated in Clausen and Žilinskas (2002), Wood (1991), and Zhang et al. (1993), simplicial partitioning might be comparatively more flexible because a wide range of partitioning schemes can be used to result in more regular geometrical shapes that are proportional to the topology of the domain. Similar to rectangular partitioning, simplicial partitioning is also convergent when nested partitioning is applied, because edge length is reduced to zero exponentially.

Practical use of Lipschitz algorithms has long been impeded by the need to specify a Lipschitz constant. By eliminating this requirement, DIRECT

and LGO enable such algorithms to deal with “black box” functions directly.

2.3 Adaptive Partitioning Methods

Adaptive partitioning approaches divide given domains into smaller subspaces non-uniformly. An efficient black box adaptive partitioning approach is the Multilevel Coordinate Search (MCS) (Huyer and Neumaier, 1999), where the non-uniform partitioning is performed by introducing a partition bias that divides boxes in the vicinity of samples having better function values. Samples are collected from box boundaries leading to sample sharing by neighbor boxes. The method combines global search (splitting the boxes with large unexplored territory) and local search (splitting boxes with good function values) and tries to balance the global and local search by a device called *level*, which is a rough measure of the number of times a box has been processed.

Tang (1994) proposes an adaptive partitioning method utilizing a generalized Pareto distribution to estimate how “promising” a partition is. The search region is subdivided into a certain number of partitions. Using sampled function values from each subregion, estimators are derived to determine how “promising” each subregion is. The most “promising” subregion is then further divided resulting in a tree search type algorithm.

Partitioning approaches that are based on fuzzy assessment are also adaptive in the sense that the selection of partition to be processed is flexible. The assessment within each partition is collected by using random search techniques, such as Simulated Annealing, hill climbing methods, etc. (Boender and Romeijn, 1995), providing evidence for fuzzy assessment of partitions (Demirhan et al., 1999; Demirhan and Özdamar, 1999; Özdamar and

Demirhan, 2000). The applications of such fuzzy assessment approaches can be found in an environmental site investigation project (Demirhan and Özdamar, 2003; Özdamar and Demirhan, 2003).

Reviews and numerical comparisons of different adaptive partitioning methods can be found in Pintér (1992), and Pintér (1996) among many others and more recently in Özdamar and Demirhan (2001).

2.4 Response Surface Methods

A more recent contribution to the field of global optimization is the techniques based on Response Surface Methodology (RSM) (Jones, 1996), which are adapted from conventional methods widely used in the field of surface re-construction. RSM-based global optimization approaches can be classified based on the type of response surface and the method used to select search points. At the highest level, they can be differentiated based on whether they are non-interpolating (minimize sum of squared errors from some predetermined functional form) or interpolating (pass through all points). Among them, optimizing by interpolating is most often used for global optimization. Cubic splines, thin-plate spline, multi-quadrics, and kriging (Deutsch and Journel, 1998) are methods that interpolate a set of discrete points using a linear combination of polynomial terms.

Radial Basis Functions (RBF) (Powell, 1992) is a popular technique in imaging; it obtains a set of samples to construct a simulated surface of the objective function through interpolation. The interpolation function consists of a linear combination of polynomial terms and special “basis function” terms that account for the non-polynomial part. The original function is estimated by a weighted set of these simpler functions where the weights are

2.4 Response Surface Methods

optimized for minimizing interpolation error. When RBF is adapted to global optimization (named as RBF-O hereafter), a target value (or, estimate) for the global optimum is selected and this sets an additional condition on the interpolation problem. The next sampling point is selected so as to minimize this utility function (Gutmann, 2001). RBF-O has been extended and commercially implemented by Björkman and Holmström (2000) where numerical tests are conducted and compared with other black box partitioning methods such as DIRECT and MCS discussed above.

Jones (2001) provides an in-depth illustration of existing two-stage and one-stage RSM approaches in global optimization (where kriging and RBF are used for surface re-construction). Two-stage approaches are techniques where the next sampling location is selected based only on the optimized fit of the re-constructed surface to the given set of collected samples. The disadvantages of such approaches are illustrated on a deceptive function by using several sample selection criteria involving kriging. Since kriging enables the calculation of a confidence interval for interpolation error, these selection criteria are based on statistical lower bounds for errors, probability of improvement, or expected improvement. Single stage approaches optimize the location of the targeted function value concurrently with the parameters of RSM. Gutmann's approach (Gutmann, 2001) is classified under this category. The sequential trial of multiple target values that Gutmann proposes to use in selecting the best target value is equivalent to multi-start search methods and this feature reinforces the global component of the search and leads to convergence. One-stage RSM approaches are strongly recommended for global optimization, but it might be computationally expensive to utilize kriging as the surface re-construction method (Jones, 2001).

Efficient Global Optimization (EGO) algorithm proposed by Jones et al. (1998) is similar to the RBF-O algorithm, in the sense that it solves global optimization problems by a response surface methodology. First a model function is fitted to the data collected by evaluating the objective function at a number of initial points. Then a utility function is used to determine the new point where the objective function should be evaluated. In EGO, a nonlinear stochastic process model, the DACE (Design and Analysis of Computer Experiments) predictor, is fitted by use of nonlinear regression. EGO balances between global and local search by choosing the new point where the objective function should be evaluated as the one which maximizes an expected improvement utility function. Jones et al. (1998) indicate that this methodology is especially efficient for modelling the nonlinear, multi-model functions that often occur in engineering.

2.5 Interval Methods

Interval methods manage partitions based on the information provided by inclusion functions which cover the actual ranges of objective and constraint functions (Ratschek and Rokne, 1988; Hansen, 1992; Ratschek and Rokne, 1995). They use interval arithmetic to solve optimization problems and adopt box partitioning approach which matches the definition and operations of interval arithmetic. Various software packages have been developed to support interval arithmetic; among them, PROFIL/BIAS (Knüppel, 1994) and XSC (Hammer et al., 1995) are widely used.

The advantage of interval methods is that they can provide explicit objective and constraint function ranges. This makes it possible to check the feasibility of a whole partition, which is difficult to achieve for those meth-

ods discussed above except the Lipschitz methods given accurate Lipschitz constants. However, such ranges are normally given in an overestimated way, which results in some infeasible partitions are treated as possibly feasible ones during partitioning. Usually, interval methods follow a branch and bound approach since tighter bounds can be calculated when the size of interval gets smaller. A box can be discarded (bounded) when it is proved that there are no better solutions than known solutions inside it. Normally, a box is branched into two sub-boxes from one of its variables (bisection); however, a box can also be subdivided from more than one direction simultaneously (Hansen, 1992; Csallner et al., 1999).

In interval methods, the order of boxes to be processed plays an important role; there exist different strategies for selecting the next box to be subdivided. Some strategies have long been used in interval methods, such as the strategy to select the box with best upper bound (Skelboe, 1974); select the box which has been longest in the list (Hansen and Sengupta, 1980); and select the box which has maximum width (Hansen and Sengupta, 1980) and so on. Berner (1996) studies the influence of different strategies for selecting the next box for subdivision in an interval branch and bound method for global optimization. More recently, some heuristic selection strategies, which utilize estimation of global optima, are proposed by Csendes (2001), Casado et al. (2001a) and Casado et al. (2001b) and so on.

Apart from the order of box selection, the subdivide direction also affects the efficiency of interval approaches since each variable's contribution in the objective function is not the same. There also exist many subdivision direction selection rules. Some are quite straight forward, such as the rule to subdivide from the widest edge (Moore, 1966; Skelboe, 1974; Ratschek and

Rokne, 1988). Some utilize the information of gradient, such as the rule of Hansen and Walster (Hansen, 1992) and the rule of Ratz (Ratz, 1992; Ratz and Csendes, 1995). In interval methods, the approach that selects the best box and splits from the widest edge is most used and usually serves as a benchmark approach.

Certain accelerating devices are introduced to speed up the convergence of interval methods. The midpoint test, which evaluates the middle point of a box, is used to discard the boxes whose upper bounds are less than the midpoint's function value. The monotonicity test, whose description can be found in Ratschek and Rokne (1988) or Hansen (1992), assumes that the given function is continuously differentiable. The box can be abdicated when this test is satisfied. However, this case is only true when dealing with unconstrained optimization. For constrained optimization, this test alone is not sufficient to discard a box. Moreover, the assumption of continuously differentiable cannot always be met in practice.

Although theoretically interval methods have no difficulties dealing with constrained optimization, the reports in constrained optimization using interval analysis are relatively few compared with those in unconstrained optimization. Some of them can be found in Wolfe (1994), where both inequality and equality constraints are treated; in Kearfott (1996b) where only equality constraints are considered, inequality constraints are converted into equality ones by adding some slack variables; and in Benyoub and Daoudi (2001) and Markót et al. where inequality constraints are incorporated and some heuristic adaptive multi-section rules are studied, among others.

It should be noted that the number of boxes increases exponentially when using interval analysis if no boxes can be discarded. This, on one hand, needs

more efficient box disposal procedures; and on the other hand, more powerful box management schemes and computations. Since interval methods follow branch and bound approach, parallel computation can be easily applied to them. Two problems concern the parallel performance: (a) how the boxes are managed; and (b) how load is balanced.

Master/Slave architecture, such as Henriksen and Madsen (1992), has the advantage of efficiently utilizing available information which makes no box handled unnecessarily. However, the disadvantage is that the master processor might become a bottleneck for the whole computation model, especially when the number of processors gets large. It seems reasonable for each processor to store a fraction of the boxes needed to be processed. Different load balancing mechanisms have been proposed. Such as, Benyoub and Daoudi (2001), propose an approach based on synchronization and uniform redistribution of boxes; Ibraev (2001) proposes a challenge leadership model where each processor has the chance becomes the “leader” when it finds better solution, the load balancing is done between the leader and idle processors; Gau and Stadtherr (2001) provide a stage-wise asynchronous work load information exchange strategy for dynamic load balancing.

2.6 Other Methods

Apart from the deterministic approaches introduced so far, there are also stochastic methods which are “derivative free”. Random search methods are such examples; different strategies can be combined with the basic idea of random sampling. Controlled Random Search (CRS) algorithm (Price, 1977) uses random generated points to form a simplex and based on new sample information to update the simplex. Adaptive Stochastic Search Methods, such

as Benson (1995), are based on—at least partially—on random sampling. Pure adaptive search (Zabinsky and Smith, 1992) iteratively constructs a sequence of interior points uniformly distributed within the corresponding sequence of nested improving regions of the feasible space. Improving Hit-and-run (IHR) (Zabinsky et al., 1993; Zabinsky, 1998) is a sequential random search method which at each iteration generates a candidate point uniformly distributed along a randomly chosen direction.

Simulated Annealing (Kirkpatrick et al., 1983; Ingber, 1993) utilizes the connection between statistical mechanics and multivariate or combinatorial optimization. Genetic Algorithms (GA) (Goldberg, 1989; Michalewicz, 1996) are based on the ideas of natural selection and genetic reproduction mechanism.

There are lots of other examples, a common characteristic of such methods is that they all incorporate random factors. As a result, different runs may give different results.

2.7 Summary

In this chapter, methods that do not utilize or can be applied without the derivative information for global optimization are reviewed. The methods listed here are by no means exhaustive and this review is not intended to cover all such methods. Most of them can be used for “black box” problems, such as direct search methods, DIRECT, LGO, MCS, RBF-O, random search methods. However, they have difficulties when dealing with constraints. On the other hand, interval methods can easily handle both objective and constraint functions. But they either need explicit expressions of objective and constraint functions, or at least need to know the inclusion functions which

2.7 Summary

cover the range of real functions of the model to be optimized (in this case, the model is treated as “black box”). Furthermore, interval methods have the advantage of providing verified solutions and knowing how closely the solutions approximate the “true” optima, which is very useful when such information is wanted.

Chapter 3

Triangulated Optimization

In this chapter, motivation and theoretical background of TRIOPT are first introduced; followed by simplex assessment and partitioning strategies; main steps and operations are illustrated and algorithm description of TRIOPT is presented. Numerical studies are conducted and results are compared with several well-known “black box” approaches. The parallel computation model of TRIOPT is implemented and its performance is analyzed.

3.1 Motivation

For unconstrained optimization problems with lower and upper bounds over variables, it seems almost intuitive to select the hypercubes (hyper-boxes) as the elementary covering subsets since the whole search area is a hypercube. For instance, box partitioning is the accepted scheme in interval methods used for global optimization because they correspond to the definition and operations in interval arithmetic (Hansen, 1992; Ratschek and Rokne, 1995). However, as indicated in Clausen and Žilinskas (2002), Wood (1991), and Zhang et al. (1993), simplicial partitioning might be comparatively more flex-

Figure 3.1: An illustration of splitting efficiency in 3D.

ible than box partitioning because a wide range of partitioning schemes can be used and in the sense that the number of new partitions obtained per additional sample taken. Define splitting performance, P_s , as follows,

Definition 3.1. *The **splitting performance**, P_s , is the ratio of the number of new partitions to the number of new vertices required, that is,*

$$P_s = \frac{\text{No. of new partitions}}{\text{No. of new vertices required}}.$$

Assuming that a sample should be collected from each new vertex (to represent the whole partition), then, one new vertex can split a 3D simplex into four new partitions by selecting one point inside, or split the simplex into two by selecting a point on one of the edges, while four new samples are generated by bisecting a box in 3D. The idea is illustrated in Figure 3.1.

In general, 2^{n-1} new samples are required for bisecting a box in n D, whereas this number is 1 in simplicial partitioning regardless of dimension. Furthermore, one can use interior splitting where a new sample is selected from inside the simplex. This results in $n + 1$ new simplices, with each child simplex formed by connecting the new vertex to each of the $n + 1$ facets of the simplex. Hence, P_s becomes $(n + 1)/1$, $2/1$, $2/2^{n-1}$ for interior simplex splitting, on-edge splitting and box bisection, respectively. Of course, it may be the case where samples are collected from location other than partition

Figure 3.2: A GIS Triangulated Irregular Network (TIN).

vertices. The splitting performance of such approaches depends on the proposed sampling schemes. By executing simplicial partitioning, TRIOPT takes advantage of the high splitting performance.

Another observation comes from the Triangulated Irregular Network (TIN) in Geographic Information System (GIS). Figure 3.2 shows a typical TIN surface plot taken from Church (2002). TIN contains a network of irregularly spaced triangles where areas of high-relief (the areas which are more important to the observer) contain a higher density of small triangles while areas of low-relief are represented by larger triangles. This approach realistically represents the topology of complex geographical surfaces while using as few data samples as possible. The same idea can be applied in global optimization since normally only areas with high value (for maximization) or low value (for minimization), that is, *high-relief areas*, are interesting. It is also reasonable that “good” samples should be favored since they might represent a high-relief area. If a method can appropriately deal with the density of sampled points over the search space, function evaluations may be saved when a global optimization problem is solved.

3.2 Simplex Assessment

3.2.1 Function Transformation

In TRIOPT, function values of samples are transformed in order to eliminate the influence of sign and achieve clearer sample discrimination. The samples are sorted in descending order of $f(\mathbf{x})$ and a threshold value t is selected from the sorted list near current maximum function value $f_{\max}$ in $\mathcal{V}$, which is the set of currently available samples. t is equal to a function value corresponding to a sample having a function value near $f_{\max}$. The transformation function $\ddot{f}$ classifies function values based on parameter t where $f(\mathbf{x}) \geq t$ are treated as peaks and $f(\mathbf{x}) < t$ as valleys. $\ddot{f}$ is calculated by using dynamic parameters: current maximum function value $f_{\max}$, current minimum function value $f_{\min}$ and threshold t . The three parameters are updated according to the currently available sample set $\mathcal{V}$. All $f(\mathbf{x})$ are transformed and mapped onto the $[0.1, 1.0]$ interval using the following formula.

$$\ddot{f}(\mathbf{x}) = \begin{cases} 0.1 + 0.4 \left(1.0 - \frac{t-f(\mathbf{x})}{t-f_{\min}} \right)^p, & \text{if } f(\mathbf{x}) < t; \\ 0.5 + 0.5 \left(\frac{f(\mathbf{x})-t}{f_{\max}-t} \right)^q, & \text{if } f(\mathbf{x}) \geq t. \end{cases} \quad (3.1)$$

Here, the deflation and inflation parameters, $p > 1.0$ and $q < 1.0$, are both positive real numbers. The basic idea is that by this monotonic nonlinear transformation, higher valued samples are favored and lower valued samples are disfavored; p and q reduce the transformed values of samples with function value below t and increase the ones above, respectively, leading to a more discriminating classification of simplices. The slope of $\ddot{f}$ changes at t and the lower bound specified in Equation (3.1) is a precaution against possible splitting degeneracies.

The nonlinear transformation of $f(\mathbf{x})$ does not change the order of samples in $\mathcal{V}$ since it is monotonic with an inflection at t . Define $\dot{f}$ as a piecewise linear transformation of $f(\mathbf{x})$ by setting the parameters p and q in Equation (3.1) to 1.0. Thus, the linear transformation function $\dot{f}$ is monotonously increasing. Further, $\ddot{f}$ has the property of including $\dot{f}$, that is, $\dot{f} \subseteq \ddot{f}$ for any $\dot{f}(\mathbf{x})$, $\mathbf{x} \in \mathcal{V}$. This holds because for any peak, $\ddot{f} > \dot{f}$ and for any valley, $\ddot{f} < \dot{f}$. Furthermore, when $f = t$, $\ddot{f} = \dot{f}$.

Remark 3.1. *Due to the relationship explained above, $\ddot{f}$ does not change the topology of f constructed from $\mathcal{V}$. The use of $\ddot{f}$ increases the attraction power of a peak $\mathbf{x}$ neighboring a valley $\mathbf{y}$ by the order of $\left(\frac{f(\mathbf{x})-t}{f_{\max}-t}\right)^{1-q} \left(1.0 - \frac{t-f(\mathbf{y})}{t-f_{\min}}\right)^{p-1}$ as compared to $\dot{f}$.*

That is, when transforming $\mathbf{x}$, the difference between $\dot{f}$ and $\ddot{f}$ is in the order of $\left(\frac{f(\mathbf{x})-t}{f_{\max}-t}\right)^{1-q}$ since $\ddot{f}$ inflates the peak $\mathbf{x}$; similarly, while transforming $\mathbf{y}$, the difference between $\ddot{f}$ and $\dot{f}$ is in the order of $\left(1.0 - \frac{t-f(\mathbf{y})}{t-f_{\min}}\right)^{p-1}$. In fact, any transformation function satisfying the above mentioned property can be used to transform f .

3.2.2 Simplex Entropy

After calculating the transformed function values $\ddot{f}$ for all $\mathbf{x} \in \mathcal{V}$, an aggregate entropy measure, E , is computed for each simplex $\mathcal{S}$ by considering the contribution of its vertices. The entropy E is an indication of potentiality of a simplex containing global optimum inside, which is calculated as:

$$E = \begin{cases} \frac{1}{k} \sum_{\mathbf{x} \in \mathcal{S}: \ddot{f}(\mathbf{x}) \geq \alpha} \ddot{f}(\mathbf{x}) \exp(1.0 - \ddot{f}(\mathbf{x})), & \text{if } \exists \ddot{f}(\mathbf{x}) \geq \alpha; \\ 0.0, & \text{otherwise.} \end{cases} \quad (3.2)$$

Here, each vertex $\mathbf{x}$ contributes to the entropy if $\ddot{f}(\mathbf{x}) \geq \alpha$, k is the number of vertices satisfying this condition. The higher the transformed value $\ddot{f}(\mathbf{x})$ is, the more contribution it gives to the entropy E . The cut α on $\ddot{f}$ prevents inferior vertices in a simplex from affecting the potentiality reflected by superior ones. The expression given for E is a multiplicative entropy form and is adapted from Pal and Pal (1989).

3.3 Partitioning Scheme

3.3.1 Initial Partition

The initial partition over $\mathcal{D}$ is constructed by triangulating an initial set of available samples $\mathcal{V}$, if such a set exists. Otherwise, a sample is collected from each vertex of $\mathcal{D}$ and one from its midpoint. The flexibility of partitioning on an arbitrary set of samples becomes an advantage for TRIOPT in black box optimization problems where a set of physical experiments is already available. However, it should be noted that if the initial sample set does not contain all vertices of the hypercube $\mathcal{D}$, the whole search domain cannot be covered. The latter necessity restricts the method to low dimensional problems.

Triangulation plays an important role in image re-construction, Geographical Information Systems (GIS), Computer Aided Geometric Design (CAGD), Finite Element Methods (FEM), CAD/CAM, numerical analysis, geology, meteorology and approximation theory. Various applications of triangulation can be found in Yu et al. (2001), Lattuada and Raper (1996), Schumaker (1993), Bern and Eppstein (1992), Cuillière (1998), to name a few. Many algorithms and methods have been proposed for triangulation,

3.3 Partitioning Scheme

Figure 3.3: Delaunay vs. non-Delaunay triangulation.

Aurenhammer (1991) and Su and Drysdale III (1995) present good surveys on different approaches. Here, the Quickhull algorithm (Barber et al., 1996), which is able to generate Delaunay triangulation on $\mathbb{R}^n$, is adopted.

Delaunay triangulation has the property that the interior of a simplex's (hyper-)circumcircle contains none of other points except the vertices of the simplex itself. This is an advantage of Delaunay triangulation to generate angle optimal simplices, which indicates the simplices are more regular. Figure 3.3 illustrates the difference between a Delaunay triangulation and a non-Delaunay triangulation of the same four points in 2D.

3.3.2 Partitioning Strategies

Once the initial partitioning is done, the entropy E_j of each simplex $\mathcal{S}_j$ can be calculated. Simplices are selected to be split when their entropies $E_j \geq \beta$, here β is a user defined parameter in the unit interval. There are two partitioning strategies proposed in TRIOPT: *interior-simplex splitting* and *longest-edge splitting*.

An interior-simplex splitting is carried out by selecting a point $\mathbf{x}_{\text{new}}$ inside the simplex $\mathcal{S}_j$ and splitting the parent simplex into $n+1$ new simplices. Each child simplex is formed by connecting the newly generated point $\mathbf{x}_{\text{new}}$ to each of the $n+1$ facets. Such an operation in 3D is depicted in Figure 3.1(a).

3.3 Partitioning Scheme

The location of the point $\mathbf{x}_{\text{new}}$ is selected by taking the weighted average over transformed function values of the relevant vertices. The mathematical formula for calculating $\mathbf{x}_{\text{new}}$ is given as follows:

$$\mathbf{x}_{\text{new}} = \sum_{\mathbf{x} \in \mathcal{S}_j} \left[\mathbf{x} \cdot \frac{\ddot{f}(\mathbf{x})^r}{\sum_{\mathbf{x} \in \mathcal{S}_j} \ddot{f}(\mathbf{x})^r} \right]. \quad (3.3)$$

Here, r is a non-negative constant that regulates the bias towards good vertices. The location of the new sample is basically a convex combination of the locations of all vertices using the transformed values $\ddot{f}(\mathbf{x})$ as their weights.

A longest-edge splitting is done by selecting a point $\mathbf{x}_{\text{new}}$ on the longest edge of the simplex. The location of $\mathbf{x}_{\text{new}}$ is calculated similar to interior-simplex splitting in Equation (3.3), except the weighted average is only carried out between the two vertices connecting the longest edge. Figure 3.1(b) illustrates a longest-edge splitting in 3D.

It is obvious that the most efficient way of splitting is the interior-simplex splitting, that is, by generating one new point (function evaluation) it can split the simplex into $n + 1$ new simplices. But one disadvantage of interior-simplex splitting is that it may result in very irregular simplices if applied repetitively. To avoid the formation of extremely irregular simplex, the concept *regular simplex* is defined:

Definition 3.2. Let L_u denote the shortest edge length of a simplex $\mathcal{S}$. If the length L_i of all edges of the simplex $\mathcal{S}$ satisfy $L_u \leq L_i \leq 2L_u$, then the simplex $\mathcal{S}$ is called a **regular simplex**.

The decision on the splitting strategy for a given simplex is made according to the following rule.

Rule 3.1. *If simplex $\mathcal{S}$ is a regular simplex, partitioning is carried out by interior-simplex splitting; otherwise, longest-edge splitting is applied.*

Rule 3.1 prevents the simplices from being too irregular and acts as a balance between regularity and splitting efficiency.

3.4 The Algorithm

3.4.1 Algorithm Description

TRIOPT can be used as a solely global search method, or it can incorporate some local search methods to explore areas around good points found by the algorithm. The Nelder-Mead local search method is very suitable to be included in TRIOPT since both methods use simplex as the basic algorithmic element. A brief introduction of TRIOPT with Nelder-Mead as local search method is given below, of course, the Nelder-Mead method can be eliminated if local search is not wanted.

After TRIOPT finishes the initial partition, a working simplex list $\mathcal{L}_w$ including all simplices is generated and all simplices' entropies are calculated. Then, an arbitrary simplex $\mathcal{S}_c$ is selected and removed from the list $\mathcal{L}_w$. If its entropy $E_{\mathcal{S}_c}$ is greater than the pre-specified entropy cut, β , then it is re-partitioned, the resulting child simplices are inserted to a temporary list $\mathcal{L}_t$, the new vertex $\mathbf{x}_{\text{new}}|\mathcal{S}_c$ obtained during re-partitioning is added to the set $\mathcal{V}$. Otherwise, the simplex $\mathcal{S}_c$ is inserted into $\mathcal{L}_t$ without partitioning. This step is repeated until all $\mathcal{S}_j \in \mathcal{L}_w$ are processed. Then for each simplex $\mathcal{S}_i$ in the temporary list $\mathcal{L}_t$, its hyper-volume $H_{\mathcal{S}_i}$ is calculated. If $H_{\mathcal{S}_i}$ is lower than the hyper-volume tolerance level, δ , $\mathcal{S}_i$ is added to the set of simplices to be reported, $\mathcal{L}_p$. Furthermore, its vertices that are not shared with other

Algorithm 3.1 TRIOPT

```

1: initial partition;
2:
3: while termination criteria are not met do
4:   while  $\mathcal{L}_w \neq \emptyset$  do
5:     select a simplex from  $\mathcal{L}_w$  as  $\mathcal{S}_c$  and delete it from the list;
6:     if  $E_{\mathcal{S}_c} \geq \beta$  then
7:       split  $\mathcal{S}_c$ ;
8:       insert the child simplices into  $\mathcal{L}_t$ ;
9:     else
10:      insert  $\mathcal{S}_c$  into  $\mathcal{L}_t$ ;
11:     end if
12:   end while
13:
14:   while  $\mathcal{L}_t \neq \emptyset$  do
15:     select a simplex from  $\mathcal{L}_t$  as  $\mathcal{S}_i$  and delete it from the list;
16:     if  $H_{\mathcal{S}_i} < \delta$  then
17:       insert  $\mathcal{S}_i$  into  $\mathcal{L}_p$ ;
18:       if no Nelder-Mead search activated with best vertex of  $\mathcal{S}_i$  then
19:         use  $\mathcal{S}_i$  to do Nelder-Mead local search;
20:       end if
21:     else
22:       insert  $\mathcal{S}_i$  into  $\mathcal{L}_w$ ;
23:     end if
24:   end while
25:   update  $f_{\max}$ ,  $f_{\min}$ ,  $t$ ;
26: end while

```

simplices are removed from the set $\mathcal{V}$. When this happens, the Nelder-Mead method is activated taking $\mathcal{S}_i$ as a starting simplex if its best vertex has not already been a source in previous Nelder-Mead activations. If $H_{\mathcal{S}_i} \geq \delta$, it is simply inserted directly to the list $\mathcal{L}_w$ so that it can be evaluated in the next assessment cycle. Once the temporary list $\mathcal{L}_t$ is depleted, the values of $f_{\max}$, $f_{\min}$ and t are updated. This triggers a new assessment cycle and the procedure continues until either there are no more simplices to assess or a termination criterion is satisfied. In engineering application problems, this

termination criterion might be a certain minimum improvement in the best solution obtained in the current assessment cycle as compared to the previous one, or, a given number of system simulations (function evaluations), or an aspiration level to be achieved for system performance. The algorithmic description of TRIOPT is given in Algorithm 3.1.

In any assessment cycle, if none of the simplex entropy levels exceeds β , then, TRIOPT may be entrapped into an infinite loop. In this case, the simplex with the maximum E_{S_j} is partitioned regardless of its entropy value. The latter policy helps the algorithm to move on since new vertices are introduced and eventually the topology of available simplices and their entropies change.

3.4.2 Complexity Analysis

For general non-convex global optimization problems, it can be shown that the problems belong to NP-complete class in terms of complexity (Vavasis, 1995). Since the problems dealt by TRIOPT are black box ones, it can be assumed that they are even harder to solve. This subsection tries to analyze the algorithm complexity in terms of samples collected during the search.

To partition a simplex, a new sample is collected either from the interior or from the longest edge of the simplex; therefore, the simplex is split into either $n + 1$ or 2 new simplices, where n is the dimension of the problem under investigation. If none of the new simplices can be disposed of, then the net list length increase on $\mathcal{L}_w$ is either n or 1 since the parent simplex is to be removed from the list. In the worst case where no longest-edge split is performed and no new simplex is disposable, the maximum length of list $\mathcal{L}_w$ is mn , where m is the number of samples collected so far. Since n is the

dimensionality of the problem which is a fixed number, it can be concluded that the algorithm's data management complexity is in the order of $O(m)$.

3.4.3 Convergence Analysis

Suppose the set of near-optimal solutions, ω , is composed of a small hull of solutions enclosing $\mathbf{x}^*$. Let the size of the basin of attraction be denoted as $m(\omega)$.

An algorithm is defined to be *convergent* if it has the capability of locating any $\mathbf{x} \in \omega$.

Theorem 3.1. *TRIOPT is a convergent algorithm given that it is not terminated by any criterion other than the simplex hyper-volume tolerance, $\delta \leq m(\omega)$.*

Properties that make the algorithm convergent are as follows.

- i. Since the simplex hyper-volume tolerance $\delta > 0$, the number of iterations required to move away from a local basin of attraction is finite;
- ii. By partitioning at least one simplex in each iteration and remove those simplices whose hyper-volume are less than δ , the maximum simplex hyper-volume $m_{\max}$ in the list $\mathcal{L}_w$ approaches the hyper-volume tolerance δ and eventually, all simplices are removed from the working list, $\mathcal{L}_w$;
- iii. Suppose t and t' are threshold values calculated just before and just after an iteration where a simplex reaches the size tolerance δ and its unshared vertices are removed. Then, $t' \leq t$. Let κ be the number of removed simplices holding local optima, f^0 be the maximum value

3.4 The Algorithm

among the vertices of the simplex containing $\mathbf{x}^*$ and τ be the number of simplices holding local optima (as reflected by $\mathcal{V}$) whose maximum sample value exceeds f^0 . As $\kappa \rightarrow \tau$, $t' \rightarrow f^0$ in a finite number of iterations, because t' is reduced in steps defined by the function values of samples in $\mathcal{V}$. In the worst case, even if the simplex containing $\mathbf{x}^*$ does not contain a vertex in ω , at this point, it becomes eligible for nested partitioning, and that eventually leads to a convergent sequence of samples.

Indeed, in each iteration several simplices are split according to their entropies and the newly generated simplices are smaller than their parent simplices, and at the same time those simplices with a size $m(\mathcal{S})$ less than δ are removed from the simplex list $\mathcal{L}_w$, this ensures that the list $\mathcal{L}_w$ eventually becomes empty. Moreover, different splitting schemes against too narrow splits guarantee that the simplex containing any point $\mathbf{x}$ shrinks sufficiently fast during splitting.

If the number of simplex splits is allowed to go to infinity, i.e., the tolerance level δ goes to zero, TRIOPT is guaranteed to converge to the global optimum if the objective function is continuous — or at least is continuous in the neighborhood of a global optimum. This follows from the fact that the set of points sampled by TRIOPT forms a dense subset of the search space if the tolerance level δ goes to zero. That is, given any point $\mathbf{x} \in \mathcal{D}$, and $\epsilon > 0$, TRIOPT will eventually generate a simplex $\mathcal{S}_j$ whose hyper-volume is less than δ with one of its vertices within the distance ϵ from $\mathbf{x}$.

These properties give the following convergence theorem for TRIOPT:

Theorem 3.2. *Suppose that the global maximization problem stated in Equation 1.1 has a solution $\mathbf{x}^*$, and that $f(\mathbf{x}) \rightarrow \mathbb{R}$ is continuous in a neighbor-*

hood of $\mathbf{x}^*$, and let $\varepsilon > 0$, $\delta' > 0$. Then, there exists a positive real number δ such that for any $\delta < \delta'$, the algorithm will eventually find a point $\mathbf{x}'$ with $f(\mathbf{x}') > f(\mathbf{x}^*) - \varepsilon$; i.e., the algorithm converges if $\delta' \rightarrow 0$.

The worst case of “eventually” in the above theorem is reached when all simplices have been removed from the simplex working list $\mathcal{L}_w$; and in this case, the search becomes equivalent to exhaustive search.

3.5 Numerical Experiments

3.5.1 Test Problems

The test problems consist of both theoretical problems and engineering applications. The theoretical problems are well-known in the literature and they are difficult to solve due to their particular topologies. Two functions are haystack types (Spike and Rastrigin), one is non-differentiable but smooth (Crescent), one has a flat surface that contains global and local optima (Six-Hump), some functions are symmetric with no trend towards the global optimum (Ackely, Easom and Griewank), and two are deceptive (G & P and Schwefel). Since TRIOPT samples the vertices and the midpoint of the search domain $\mathcal{D}$ to form the initial partition, boundaries of some symmetric test problems (i.e., the global optimum lies at the midpoint) are shifted so that the optimal solutions are not found trivially.

Among the theoretical test functions, Six-Hump is a problem where global and local optima are on a flat surface, the difference between global optima and local ones is small; there are six local optima in the search domain, two out of them are global ones. G & P is a deceptive problem with very large function range, the global optimum and three local optima are on a flat

3.5 Numerical Experiments

surface. Ackley is a highly multi-modal test problem with evenly distributed optima, the cosine part of the function contributes to its multi-modality. The Spike is similar to the Ackley problem with innumerable local optima located in a “haystack type” way, the difference between the global optimum and the second best solution is quite small; further, the basin of attraction for the global optimum is very narrow. Schwefel is a deceptive function where global optimum is located near the domain’s corner, three second best solutions are very far away from the global optimum together with a lot of other local optima that render the difficulty of the problem. Easom is a unimodal test problem with a very flat surface outside the basin of attraction. Crescent is a non-differentiable test problem, there is a circular loci on which a singular global optimum lies on a crater-like basin of attraction, and it is smooth elsewhere. Griewank is a problem with many widespread evenly distributed local optima. Rastrigin is a highly multi-modal, evenly spaced haystack type of test function, with local and global optima located on a spherical surface with cosine modulation to produce the multi-modality.

The engineering applications include a QFT bounds calculation problem (ANVCS), an optimal shape design problem (Truss structure), two data fitting problems (Weeds and Palmer 4), a composite pressure vessel design problem (Pressure vessel) and a set of protein folding problems (Protein folding) which consists of 8 problems with different combinations of two amino acids. Among them, ANVCS is a smooth problem but it is discontinuous in the vicinity of the global optimal solution. The optimal solution lies on the edge of the discontinuity. The Truss structure has a very flat surface near the global optimum that might lead to wasted function evaluations. Palmer 4 data fitting problem involves even degree polynomials, hence its surface is

3.5 Numerical Experiments

Table 3.1: Summary of test problems for TRIOPT.

<i>Problem</i>	<i>Dimension</i>	<i>Reference</i>
Six-Hump	2	GEATbx ^a
G & P	2	GEATbx ^a
Ackley	2	Bäck (1996)
Spike	2	Michalewicz (1996)
Schwefel	2	Schwefel (1981)
Easom	2	GEATbx ^a
Crescent	2	Mäkelä and Neittaanmäki (1992)
Griewank(2)	2	Törn and Žilinskas (1989)
Griewank(3)	3	Törn and Žilinskas (1989)
Griewank(4)	4	Törn and Žilinskas (1989)
Rastrigin(2)	2	Törn and Žilinskas (1989)
Rastrigin(3)	3	Törn and Žilinskas (1989)
Rastrigin(4)	4	Törn and Žilinskas (1989)
Rastrigin(5)	5	Törn and Žilinskas (1989)
ANVCS	2	Nataraj and Sheela (2002)
Truss structure	2	Pownuk (2000)
Weeds	3	CUTEr ^b
Palmer 4	4	CUTEr ^b
Pressure vessel	3	Krikanov (2000)
Protein folding	3	Stillinger et al. (1993)

^a GEATbx: Genetic and Evolutionary Algorithm Toolbox for Matlab. http://www.systemtechnik.tu-ilmenau.de/~pohlheim/GA_Toolbox/fcnindex.html.

^b CUTEr: A Constrained and Unconstrained Testing Environment, revisited. <http://cutter.rl.ac.uk/cute-www/problems.html>.

expected to be more irregular as compared to the Truss structure and ANVCS. Protein folding problems are highly multi-modal and therefore quite difficult to solve, so is the Pressure vessel design problem. The engineering applications that are included in the test set aim to show the solution performance of TRIOPT in comparison with other approaches. While the theoretical test problems are twice differentiable in most cases some of the engineering applications are not, and therefore more difficult to solve.

Table 3.1 gives a summary of the dimension and reference of each test problem. Detailed description of test problems can be found in Appendix A.

3.5.2 Methods for Comparison

TRIOPT is compared with several global partitioning optimization methods. They are briefly introduced below.

DIRECT

The first one is DIRECT (Jones et al., 1993). The commercial implementation, the module *glbSolve* in the TOMLAB® (<http://www.tomlab.biz>) software package, is utilized here. Similar to TRIOPT, DIRECT has an adaptive concurrent partitioning scheme that is based on rates of change in box size and function value. Multiple boxes are trisected simultaneously if they are non-dominated with respect to the two criteria of mid-point function value and box width. This approach is based on Shubert's One-Dimensional Lipschitzian method in Shubert (1972), but does not need a Lipschitz constant. The method is efficient as it creates three new boxes with two new points. The third point becomes the mid-point of the parent box.

VTD-IF

Another implementation of DIRECT, which is written in a non-commercial FORTRAN code, is VTDIRECT (<http://www.cs.vt.edu/ltw/VTdirect.ar.gz>). Based on this, Carter et al. (2001) propose an approach where DIRECT can be implemented either as a stand alone partitioning algorithm or as the first stage of a two-stage procedure. The two-stage procedure executes VTDIRECT until a given number of function evaluations. Then the solution obtained by VTDIRECT in the first stage becomes a starting point for the Implicit Filtering algorithm (IF) that is basically a local search algorithm based on projected quasi-Newton steps calculated with finite differences. IF

code can be downloaded from <http://www4.ncsu.edu/~ctk/iffco.html>.

The approach is denoted as “VTD-IF” here.

MCS

Multilevel Coordinate Search (MCS) proposed by Huyer and Neumaier (1999) is similar to DIRECT, however, rather than taking samples at the middle points, MCS takes samples at the boundary of the boxes. A golden section type of splitting is carried out by generating two new points resulting in four boxes. Splitting can be carried out in two ways by selecting the coordinate with the least number of historical splits or by the coordinate that will produce the maximum expected gain. In the latter case, two points and the base point are used to form a quadratic interpolation of the coordinate and the minimizer is taken as the splitting points. MCS has an important parameter that affects results. This parameter is the maximum depth of the tree, $s_{\max}$. It is necessary to increase this parameter when MCS cannot converge to the global optimum. MCS can also be accompanied by a local search, applied after a box is split $s_{\max}$ times. The implementation of MCS is carried out by the MATLAB[®] code provided in <http://www.mat.univie.ac.at/~neum/software/mcs/>.

LGO

The Lipschitz(-Continuous) Global Optimizer LGO (Pintér, 1997) which incorporates derivative free random sampling techniques within a global partitioning algorithm is also included in the methods for comparison. The commercial software for LGO is utilized here (Pintér, 2003) with the two random search options: Global Adaptive Random Search (GARS) and Multi

Start Global Random Search (MS).

PDS

The simplicial Parallel Direct Search (PDS) (Dennis Jr. and Torczon, 1991) is also included in the experiments. The C++ code can be downloaded from the site <http://hpcrd.lbl.gov/~meza/projects/opt++/OPT++doc/html/classOptPDS.html> (Meza, 1994). It is applied as a stand-alone application to illustrate the performance of a convergent simplicial direct search algorithm in test problems. PDS involves basic simplicial moves (reflection, expansion, contraction) that are applied in a specific pattern acting like a line search that re-traces its steps. PDS guarantees strictly monotonic decrease of sample values until a stationary point is reached. It is designed to construct n parallel line searches that results in a more effective exploration strategy.

3.5.3 Implementation Details

Performance Assessment

The assessment for performance is measured by the number of function evaluations that a method takes to converge to an ε -optimal solution that is within a small percentage below the global optimum solution, the formula is given below:

$$\begin{cases} \frac{|f^* - f|}{|f^*|} < \varepsilon & \text{if } f^* \neq 0.0 \\ |f^* - f| < \varepsilon & \text{if } f^* = 0.0 \end{cases} \quad (3.4)$$

The relative error ε is set to 1.0% and 0.01% in the numerical experiments. An exception is made for the Protein folding problems where the deviation

3.5 Numerical Experiments

from the best function value obtained over all methods is reported within a given number of evaluations, 100, 300, 500, 1000, 2000, respectively. This exception is due to the significantly large number of function evaluations reported for convergence (Stillinger et al., 1993).

Parameters

In the numerical experiments, TRIOPT starts with a given set of samples collected from each vertex of the hypercube $\mathcal{D}$ and one additional sample at its midpoint. The size tolerance level, δ , is set to $(\prod_{i=1}^n l_i)/8000$ where l_i is bound length of the i^{th} coordinate. The parameters (p, q) used in Equation (3.1) are set to $(3, 0.5)$. Numerical experiences in subsection 3.5.4 indicate that the results are not sensitive to different combination of p and q once the basic properties of (p, q) are not changed. The α and β cuts are fixed to 0.7 and 0.9, respectively to enable fast convergence. With such a high value set for the cut α , the parameter p has no effect on the aggregated entropy measure E , but it does affect the location of the new sample when a simplex is re-partitioned since it affects the transformed values of the vertices.

Two versions of TRIOPT are tested to show the impact of local search on the global partitioning algorithm. They are TRIOPT applied with and without Nelder-Mead calls, denoted as TRIOPT and TRI-NM, respectively. In the implementation, Nelder-Mead local search stops when the largest edge of the working simplex is less than 10^{-5} or when there is no improvement 25 times consecutively.

In the numerical tests, DIRECT is applied with its default settings since there are no specific parameters for it. In VTD-IF, VTDIRECT is executed until a certain level of percentage below the global solution is achieved or a

3.5 Numerical Experiments

given number of function evaluations are spent. Here, VTDIRECT is stopped when the best solution obtained is 30% and 20% below the known global solution, and proceed on feeding the best solution to IF until the desired accuracy level is reached; these two are marked as VTD-IF1 and VTD-IF2, respectively. In the Protein folding problems, VTDIRECT is terminated at 70% and 80% of the number of function evaluations allowed. They are also denoted as VTD-IF1 and VTD-IF2 in the tabulated results for the Protein folding problems. The parameter $s_{\max}$ in MCS is set to two values, $5n + 10$ (recommended one) and $15n + 10$, and are labeled as MCS1 and MCS2, respectively. The local search option is also activated in the test runs, marked as MCS2L, that is, the local search is implemented with MCS2 ($s_{\max} = 15n + 10$). LGO is implemented with two random search options: Global Adaptive Random Search (GARS) and Multi Start Global Random Search (MS). However, both give the same results during the test runs. In PDS, the simplex type 1 is utilized for numerical experiments.

3.5.4 Parameter Sensitivity Analysis

Before presenting the numerical results, different combinations of α and β , p and q are tested on the set of theoretical test problems to analyze the parameter sensitivity. The combinations of the parameters α and β are tested with these value pairs: (0.5, 0.7), (0.6, 0.7), (0.6, 0.8), (0.7, 0.8) and (0.8, 0.9). The parameters p and q are examined with p values of 1.5, 3.0, 5.0 and 10.0, and q values of 0.3, 0.5 and 0.8, respectively. Hence, there are 5 combinations of α and β and 12 combinations of p and q . Both sets of combinations are tested at the 1.0% and 0.01% accuracy levels.

Table 3.2 shows the results of these combinations of α and β at the 1.0%

3.5 Numerical Experiments

Table 3.2: Performance of different combinations of α and β at 1.0% and 0.01% accuracy levels.

<i>Function</i>	<i>(0.5, 0.7)</i>	<i>(0.6, 0.7)</i>	<i>(0.6, 0.8)</i>	<i>(0.7, 0.8)</i>	<i>(0.7, 0.9)</i>
<i>1.0% accuracy level</i>					
Six-Hump	101	53	53	53	53
G & P	57	42	42	42	42
Ackley	102	78	78	68	68
Spike	218	210	210	208	208
Schwefel	803	534	534	534	534
Easom	25	21	21	21	21
Crescent	6	6	6	6	6
Griewank(2)	83	70	70	57	57
Griewank(3)	634	633	633	488	488
Griewank(4)	430	429	429	416	416
Rastrigin(2)	80	90	90	90	90
Rastrigin(3)	123	109	109	109	109
Rastrigin(4)	554	1087	1087	1086	1086
Rastrigin(5)	1729	1680	1680	1680	1680
<i>0.01% accuracy level</i>					
Six-Hump	160	82	82	82	82
G & P	57	42	42	42	42
Ackley	443	408	408	408	408
Spike	244	244	244	242	242
Schwefel	940	884	884	884	884
Easom	25	21	21	21	21
Crescent	6	6	6	6	6
Griewank(2)	3353	3204	3204	668	668
Griewank(3)	634	633	633	488	488
Griewank(4)	845	759	759	731	731
Rastrigin(2)	80	90	90	90	90
Rastrigin(3)	123	109	109	109	109
Rastrigin(4)	803	1087	1087	1086	1086
Rastrigin(5)	1729	1680	1680	1680	1680

and 0.01% accuracy levels. It can be observed that in most cases, lower values of α and β incur more function evaluations except in Rastrigin(2) and Rastrigin(4) for both 1.0% and 0.01% accuracy levels. The parameters α and β control the degree of parallelism, that is, the lower α and β are,

3.5 Numerical Experiments

the greater the parallelism is. Lower values enable more simplices to be eligible for partitioning at each iteration, which consequently leads to more function evaluations. The exceptional cases of Rastrigin(2) and Rastrigin(4) might due to the fact that the greater parallelism helps to incorporate the simplices which contain global optimum at earlier stages, however, the overall performance indicates that higher values of α and β enable faster convergence in general.

Tables 3.3 and 3.4 present the results of different combinations of p and q values with α and β fixed to (0.7, 0.9) at 1.0% and 0.01% accuracy levels, respectively. It is observed that the function evaluations required for different combinations are different since the parameters p and q do affect the transformed values which are used to determine the location of new samples to be taken. However, the performance difference between different combinations is not statistically significant which indicates that although the locations of the new samples change, the simplices selected for partitioning seem to be relatively fixed. Since p and q are just transforming the samples with monotone transformation, good simplices will always have good aggregated entropies, which explains the relatively small performance difference.

The performance of different combinations is compared with that of combination $p = 3.0$ and $q = 0.5$. The function evaluations by this combination are taken as base and all other combinations' function evaluations are represented as percentile of the pair $p = 3.0, q = 0.5$. Table 3.5 presents the minimum, maximum and average percentile performance (of function evaluations over the base) of different combinations of p and q among all test instances. Although the smallest percentage is 18.97% and the highest percentage is 181.25%, the average percentages for all combinations are always

3.5 Numerical Experiments

Table 3.3: Performance of different combinations of p and q at 1.0% accuracy level.

	$(.3,1.5)$	$(.3,3)$	$(.3,5)$	$(.3,10)$	$(.5,1.5)$	$(.5,3)$	$(.5,5)$	$(.5,10)$	$(.8,1.5)$	$(.8,3)$	$(.8,5)$	$(.8,10)$
Six-Hump	53	53	53	53	53	53	53	53	53	53	53	53
G & P	42	42	42	42	42	42	42	42	40	40	40	40
Ackley	74	74	74	74	68	68	68	68	61	61	61	61
Spike	298	162	175	205	230	208	207	189	216	225	151	209
Schwefel	470	534	642	630	472	534	642	630	472	514	642	630
Easom	21	21	21	21	21	21	21	21	21	21	21	21
Crescent	6	6	6	6	6	6	6	6	6	6	6	6
Griewank(2)	72	69	68	67	71	57	67	67	71	46	67	65
Griewank(3)	350	495	539	419	397	488	413	531	697	532	440	371
Griewank(4)	754	691	492	400	646	416	577	343	630	471	457	381
Rastrigin(2)	27	27	27	27	27	90	27	27	27	85	27	27
Rastrigin(3)	103	109	109	103	103	109	109	103	103	109	109	103
Rastrigin(4)	206	1087	750	448	466	1086	749	447	466	1086	749	447
Rastrigin(5)	1864	1680	1726	1793	1164	1680	1843	1864	1583	1681	1946	1932

3.5 Numerical Experiments

Table 3.4: Performance of different combinations of p and q at 0.01% accuracy level.

	$(.3,1.5)$	$(.3,3)$	$(.3,5)$	$(.3,10)$	$(.5,1.5)$	$(.5,3)$	$(.5,5)$	$(.5,10)$	$(.8,1.5)$	$(.8,3)$	$(.8,5)$	$(.8,10)$
Six-Hump	82	82	82	82	82	82	82	82	82	82	82	82
G & P	42	42	42	42	42	42	42	42	40	40	40	40
Ackley	74	74	74	74	68	68	68	68	61	61	61	61
Spike	308	213	192	322	244	242	312	265	275	239	294	273
Schwefel	926	834	652	652	926	884	652	652	926	816	652	652
Easom	21	21	21	21	21	21	21	21	21	21	21	21
Crescent	6	6	6	6	6	6	6	6	6	6	6	6
Griewank(2)	867	618	852	838	769	668	679	573	726	882	617	646
Griewank(3)	613	495	539	564	563	488	413	531	697	532	440	417
Griewank(4)	754	691	971	739	646	731	921	447	633	773	675	443
Rastrigin(2)	27	27	69	99	27	90	69	83	27	85	58	81
Rastrigin(3)	103	109	109	103	103	109	109	103	103	109	109	103
Rastrigin(4)	742	1087	750	753	728	1086	1162	466	727	1086	1162	466
Rastrigin(5)	1934	1680	1843	1864	1245	1680	1843	1885	1689	1681	1952	1964

3.5 Numerical Experiments

Table 3.5: Percentage of evaluations of different combinations of p and q compared to $p = 3.0, q = 0.5$.

p	$q = 0.3$			$q = 0.5$			$q = 0.8$		
	<i>min.</i>	<i>max.</i>	<i>avg.</i>	<i>min.</i>	<i>max.</i>	<i>avg.</i>	<i>min.</i>	<i>max.</i>	<i>avg.</i>
<i>1.0% accuracy level</i>									
$p = 1.5$	18.97	181.25	98.13	30.00	155.29	92.63	30.00	151.44	96.97
$p = 3.0$	30.00	166.11	100.39	100.00	100.00	100.00	80.70	113.22	99.06
$p = 5.0$	30.00	120.22	97.36	30.00	138.70	97.81	30.00	120.22	93.58
$p = 10.0$	30.00	117.98	92.67	30.00	117.98	92.45	30.00	117.98	90.41
<i>0.01% accuracy level</i>									
$p = 1.5$	30.00	129.79	100.52	30.00	115.37	92.15	30.00	142.83	95.24
$p = 3.0$	30.00	108.82	93.55	100.00	100.00	100.00	89.71	132.04	101.24
$p = 5.0$	69.06	132.83	99.16	73.76	128.93	100.59	64.44	121.49	95.91
$p = 10.0$	69.34	133.06	103.04	42.91	112.20	91.49	42.91	116.90	89.90

around 100.00% with the lowest at 89.90% and the biggest at 103.04%. This suggests that although the performance of different combinations varies, the overall effort needed to solve all test instances for each combination of p and q is similar. To validate this assumption, a two-way ANOVA (Analysis of Variance) is conducted for both 1.0% and 0.01% accuracy levels. The statistical results show that there is no statistically significant difference among the average performance of different combinations of p and q , and there is no interaction between p and q when the confidence level is set to $\alpha = 0.05$.

Overall, it can be concluded that the parameters α and β affect the degree of parallelism, the lower the parameters are, the more the number of function evaluations is required. The values of the parameters p and q have limited impact on the performance, and therefore, the performance is not sensitive to the different combinations of p and q once the monotone transformation property holds.

3.5.5 Numerical Results

Tables 3.6 and 3.7 show the number of function evaluations needed to solve all test problems except the Protein folding problems within 1.0% and 0.01% tolerance below the global optimum, respectively. In the tables, “—” indicates the corresponding method cannot solve the problem within the given accuracy level. For DIRECT, this indicates that it suffers from non-smooth surfaces as it has to partition into small subregions that cover narrow basin of attractions; for VTD-IF, it means that the method cannot get away from its initial search point since IF is basically a local search method; for MCS, it implies that the maximum level of the tree is insufficient to achieve the desired accuracy. for LGO and PDS, it suggests that the methods may suffer

from the high multi-modality of the test problems.

For 1.0% accuracy, it is observed that TRI-NM performs better than the other methods in Griewank(2), Palmer 4 and Pressure vessel. DIRECT performs best in Schwefel, Crescent and Truss structure at 1.0% level, but it has a weak performance on symmetric and haystack type functions where the criterion for selecting boxes loses its discrimination power. VTD-IF performs better than DIRECT in most problems other than Rastrigin. Normally, VTD-IF2 takes more evaluations than VTD-IF1, but the chance that VTD-IF1 fails is also higher. This indicates that when VTDIRECT provides the correct location of global optimum, Implicit Filtering (IF) can quickly find it. A dramatic improvement using VTD-IF is observed on Spike. MCS seems to be highly sensitive to the parameter that defines the maximum level of tree, $s_{\max}$. Using the recommended default $s_{\max}$, MCS1 identifies the global optimum quite quickly in G & P, Ackley and Spike, but it fails to converge to the optimum in Schwefel and Crescent, Griewank(3) and (4) and all variants of Rastrigin, where smaller boxes need to be generated to capture the global optimum. MCS2 usually results in higher function evaluations than MCS1 except for the cases when it can converge to global optimum (Schwefel, Crescent, Griewank(3), Griewank(4) and Rastrigin(2)). In variants of Griewank and Rastrigin, MCS2L seems to perform best. With 1.0% accuracy, LGO performs well in Ackley, Easom, variants of Griewank. PDS outperforms other methods in G & P, Ackley and Rastrigin(2); but it cannot solve Spike and Rastrigin(3), (4) and (5). TRIOPT's performance is not very impressive but quite stable since other methods fail on at least one problem except the MCS2L, which incorporates local search. In particular, its performance in low and high dimensional Rastrigin problem is notable, all other meth-

3.5 Numerical Experiments

Table 3.6: Results for test problems with 1.0% accuracy level.

<i>Problem</i>	<i>TRIOPT</i>	<i>TRI-NM</i>	<i>DIRECT</i>	<i>VTD-IF1</i>	<i>VTD-IF2</i>	<i>MCS1</i>	<i>MCS2</i>	<i>MCS2L</i>	<i>LGO</i>	<i>PDS</i>
Six-Hump	53	43	102	47	47	5398	46	41	1329	195
G & P	42	44	101	68	91	13	23	50	1466	5
Ackley	68	50	93	54	82	18	38	33	12	11
Spike	208	335	10803	—	78	82	136	244	2482	—
Schwefel	534	1155	191	197	197	—	1064	1111	10414	771
Easom	21	27	39	28	28	33	61	38	7	89
Crescent	6	28	5	5	5	—	163	249	21	66
Griewank(2)	57	14	81	48	68	42	87	49	28	30
Griewank(3)	488	351	145	—	—	—	399	69	36	134
Griewank(4)	416	160	169	—	206	—	1264	131	55	449
Rastrigin(2)	90	59	235	232	232	—	2324	39	6261	30
Rastrigin(3)	109	87	1627	—	—	—	—	54	10438	—
Rastrigin(4)	1086	481	7479	16156	16847	—	—	72	—	—
Rastrigin(5)	1680	727	29467	52573	54117	—	—	90	—	—
ANVCS	33	32	—	—	141	—	—	69	4	200
Truss structure	6	6	5	9	9	17	26	34	12	17
Weeds	8387	143	15335	6435	6594	—	36879	93	188	5866
Palmer 4	30	19	57	60	60	2324	8624	75	93	277
Pressure vessel	113	48	10087	1790	2132	12199	19293	19293	337	801

3.5 Numerical Experiments

Table 3.7: Results for test problems with 0.01% accuracy level.

<i>Problem</i>	<i>TRIOPT</i>	<i>TRI-NM</i>	<i>DIRECT</i>	<i>VTD-IF1</i>	<i>VTD-IF2</i>	<i>MCS1</i>	<i>MCS2</i>	<i>MCS2L</i>	<i>LGO</i>	<i>PDS</i>
Six-Hump	82	53	178	80	80	5398	169	51	1336	275
G & P	42	55	191	106	105	4651	30	50	1490	5
Ackley	68	75	271	59	108	18	38	95	12	11
Spike	242	352	12423	—	—	—	1166	1522	—	—
Schwefel	884	1172	319	231	231	—	1064	1111	10414	844
Easom	21	42	95	45	45	—	62	38	9	208
Crescent	6	56	5	5	5	—	—	1743	51	66
Griewank(2)	668	59	711	—	89	—	91	58	32	30
Griewank(3)	488	376	9969	—	—	—	403	80	55	134
Griewank(4)	731	206	35059	—	—	—	1270	131	79	847
Rastrigin(2)	90	69	323	258	258	—	2432	39	9362	30
Rastrigin(3)	109	115	1947	—	—	—	—	54	10438	—
Rastrigin(4)	1086	515	9563	—	16891	—	—	72	—	—
Rastrigin(5)	1680	767	35077	—	—	—	—	90	—	—
ANVCS	80	64	—	—	152	—	—	—	4	200
Truss structure	52	52	79	27	27	70	123	49	21	138
Weeds	8387	151	15533	6449	6606	—	36879	93	278	5935
Palmer 4	139	73	905	130	130	7759	9000	89	120	1548
Pressure vessel	113	67	10131	1804	2156	12235	19471	19471	353	847

3.5 Numerical Experiments

ods either fail in this problem without local search or require a much higher number of function evaluation to converge.

In engineering problems, TRI-NM and TRIOPT perform quite well. It is observed that DIRECT and MCS without local search cannot solve ANVCS due to its discontinuity located at the global optimum. For DIRECT, if the middle point of a box falls on the wrong side of the discontinuity, the box does not have chance to be re-partitioned in the early stages of the search. On the other hand, TRIOPT encloses the discontinuity with its triangular vertices, and eliminates the effect of the vertex on the wrong side by implementing the transformation cut. The performance of TRI-NM and TRIOPT on Pressure vessel is quite impressive, DIRECT and MCS spend excessive amount of function evaluations to converge, and local search of MCS does not help too much on this problem.

Figure 3.4 summarizes the times each method performs best or second best together with the times the method fails out of the 19 test problems. It can be observed that TRI-NM performs best among all methods and MCS2L behaves as a runner-up. LGO outperforms TRIOPT in number of best and second best times, but it fails on some problems. TRI-NM, MCS2L and TRIOPT are the three methods without failing on any problem, and among them, TRIOPT is the only one without local search.

When the required accuracy is set to 0.01%, it is observed that MCS1 which is limited by a lower maximum tree level cannot converge for most of the test problems. MCS2 is also affected in convergence which indicates the tree level is not deep enough. The number of function evaluations required by DIRECT increasing extremely fast for the Griewank problems. In this refined accuracy requirement, it is observed that TRIOPT and TRI-NM

3.5 Numerical Experiments

Figure 3.4: Best and second best performed times and the number of unsolvable problems of different methods with 1.0% accuracy level.

become highly competitive as compared with other methods. Furthermore, only TRIOPT and TRI-NM do not get stuck in all test problems.

Figure 3.5 illustrates the times of best and second best and unsolvable times of each method. MCS2L holds the position of producing most times of best and second best performance at 0.01% level, but it also fails on one problem. The trend of other methods is similar to the case of 1.0% accuracy level. The difference is that at this level only TRIOPT and TRI-NM do not get stuck while other methods have at least one problem unsolvable.

In general terms, the splitting performance P_s for MCS and DIRECT are $4/2$ and $3/2$, respectively. For TRIOPT, P_s equals to $n+1$ and 2 when splitting is carried out according to interior-simplex and longest-edge splitting for nD problems. And similar to MCS, the longest-edge splitting leads to

3.5 Numerical Experiments

Figure 3.5: Best and second best performed times and the number of unsolvable problems of different methods with 0.01% accuracy level.

share of sample with neighboring simplices. Thus, the number of function evaluations needed for partitioning is small as compared to MCS and DIRECT. However, MCS only partitions one box at a time, while DIRECT has a parallel splitting strategy based on the criteria pair (function value, box width). Similar to DIRECT, TRIOPT also has a parallel strategy based on entropy. Due to its higher P_s , TRIOPT produces more partitions that may become eligible for re-partitioning, especially in problems where extremities are very close to each other in terms of function value and hence reflect no trend towards the global optimum.

In the results shown in Table 3.8 for the 8 Protein folding problems that are highly multi-modal, MCS2L is clearly superior throughout all permitted function evaluation levels. It should be noted that the performance of

3.5 Numerical Experiments

Table 3.8: Results for protein folding problems with different number of function evaluations.

	<i>TRIOPT</i>	<i>TRI-NM</i>	<i>DIRECT</i>	<i>VTD-IF1</i>	<i>VTD-IF2</i>	<i>MCS1</i>	<i>MCS2</i>	<i>MCS%L</i>	<i>LGO</i>	<i>PDS</i>
<i>100 evaluations</i>										
Average % dev.	4.133	3.595	21.750	12.644	16.481	13.800	21.750	0.0	5.424	2.891
Maximum % dev.	19.408	13.628	92.331	77.624	83.246	37.614	93.331	0.0	14.825	8.258
<i>Best/second best</i>	4	6	0	4	4	0	0	8	0	2
<i>300 evaluations</i>										
Average % dev.	0.041	0.005	3.461	1.030	3.489	4.906	12.197	0.0	4.448	0.333
Maximum % dev.	0.102	0.025	10.319	8.192	27.580	16.217	31.132	0.0	14.848	0.947
<i>Best/second best</i>	0	8	0	2	0	0	0	8	0	0
<i>500 evaluations</i>										
Average % dev.	0.057	0.0	3.202	0.037	0.031	3.730	9.250	0.0	0.0	0.074
Maximum % dev.	0.195	0.0	10.151	0.296	0.247	15.707	31.132	0.0	0.0	0.132
<i>Best/second best</i>	3	8	2	5	7	0	0	8	8	1
<i>1000 evaluations</i>										
Average % dev.	0.034	0.0	1.522	0.024	0.024	3.723	5.300	0.0	0.0	0.009
Maximum % dev.	0.100	0.0	4.567	0.191	0.188	15.707	14.530	0.0	0.0	0.018
<i>Best/second best</i>	4	8	2	7	7	0	3	8	8	4
<i>2000 evaluations</i>										
Average % dev.	0.029	0.0	1.512	0.0	0.0	3.729	2.986	0.0	0.0	0.0
Maximum % dev.	0.100	0.0	4.531	0.0	0.0	15.707	11.781	0.0	0.0	0.0
<i>Best/second best</i>	6	8	2	8	8	0	4	8	8	8

all methods is given in percentage deviation from the best solution found in that particular function evaluation level and hence, at different function evaluation levels a method might seem to deteriorate as compared to its previous performance. Among methods other than MCS2L (that can be used as a reference point for bench marking others), it is observed that LGO finds those same best solutions at the level of 500 function evaluations. However, its percentage deviation is quite high at 300 function evaluations. TRI-NM achieves all best solutions at 1000 function evaluations, whereas VTD-IF and PDS achieves the same performance at 2000 function evaluations. However, it should be noticed that TRI-NM provide solutions very close to the benchmarks starting from 300 function evaluations level. Among all stand alone partitioning methods, TRIOPT is superior at all function evaluation levels with a worst performance (indicated as Maximum % dev.) that is quite satisfactory as compared to other methods.

3.6 Parallel Implementation

3.6.1 Brief Introduction

Before introducing the parallel model of TRIOPT, some algorithmic descriptions of communication between processors are given and parallel performance measurement is defined.

Message Passing Routines

Throughout this research, the message sending and receiving operations in parallel implementations are carried out by Message Passing Interface (MPI). MPI is a standard library interface for writing parallel programs. It was

3.6 Parallel Implementation

developed in two phases by an open forum of parallel computer vendors, library writers, and application developers (Message Passing Interface Forum, 1994).

Here, the message passing operations are described as pseudo codes for conciseness and written in small capital letters to differentiate them from other operations. The following communication operations are defined:

SEND_MESSAGE (content, destination, tag): This operation sends the message “content” to the “destination” processor. The “tag” instructs the “destination” processor what kind of operation it should take. In **TRIOPT**, the contents can either be a simplex $\mathcal{S}$ or nothing, the tag can be one of **SIMP**, **LOCAL**, **IDLE** or **QUIT**. In interval analysis, the contents is an interval (a box), the tag might be **BOX**, **IDLE** or **QUIT**.

RECV_MESSAGE (content, source, tag): The receiving operation is the counterpart of sending operation, the “source” indicates from which processor the message is sent. When the “source” is not specified, it means this operation can receive from any processor if there is a matched message arrival. The “content” and “tag” are similar as they are in **SEND_MESSAGE**.

ALLGATHER (content): This operation makes each processor collect the “content” on each processor into a local array.

The messages can be sent and received either in a blocking way or non-blocking way. Blocking sending and receiving do not return until the operations are finished, while non-blocking returns immediately which makes process communication overlap process computation. However, non-blocking sending and receiving cannot guarantee the message passing operations are

completed when they return. When a non-blocking message passing operation is adopted, an “NB_” is used as a prefix for the operation, and flags F_s and F_r are used for detecting the complement of sending and receiving operations, respectively. Thus, the sending and receiving operations are: NB_SEND_MESSAGE (content, destination, tag, F_s) for non-blocking sending and NB_RECV_MESSAGE (content, source, tag, F_r) for non-blocking receiving.

Parallel Performance Measurement

The concepts of *speedup* and *efficiency* are defined following definitions commonly used in parallel processing for parallel performance measurement:

Definition 3.3. Speedup, $S(p)$ is the ratio of the solution time T_{serial} for a problem with the serial code executed on a single processor, to the solution time $T_{\text{parallel}}(p)$ with the parallel algorithm executed on p processors.

$$S(p) = \frac{T_{\text{serial}}}{T_{\text{parallel}}(p)}.$$

Definition 3.4. Efficiency, $E(p)$ is the ratio of speedup $S(p)$ to the number of processors p :

$$E(p) = \frac{S(p)}{p}.$$

Efficiency provides a way to measure the performance of an algorithm independently from the power of processors on the particular computer.

Definition 3.5. Linear speedup is the case when a parallel algorithm executed on p processors runs p times faster than the serial algorithm runs on a single processor.

Linear speedup indicates that the efficiency is equal to 1. Correspondingly, there exist *sub-linear speedup* with efficiency less than 1 and *super-linear speedup* with efficiency greater than 1. Sub-linear speedup is common due to the presence of sequential segments of code, delays for processor synchronization, overhead in communication and so on. Super-linear speedup, which would be greatly welcomed, is rather unusual. It means that the parallel model explores the solution set in a more efficient way.

3.6.2 The Parallel Model

In TRIOPT, all simplices with entropy $E \geq \beta$ are selected for re-partitioning in parallel, hence the number of simplices re-partitioned in parallel is dynamic and depends on global information $(f_{\max}, f_{\min}, t)$ updated during the execution of the search. There are two occasions where these parameters might change. The first one takes place when a simplex size reaches the tolerance level, δ and all vertices of the simplex that are not shared by others are discarded from $\mathcal{V}$. Hence, $f_{\max}$, $f_{\min}$ and t might be modified to result in readjusted $\check{f}$ values. The second occasion is encountered regularly when new sample, $\mathbf{x}_{\text{new}}$ changes global information. Both occasions lead to the re-definition of peaks and valleys, however, the first occasion is important for convergence. In particular, if $f_{\max}$ is reduced with the removal of a small simplex, previously inferior simplices might now become peaks to be re-partitioned. In the limit, the algorithm converges to the simplex that contains the global optimum even if its vertices do not initially indicate a promise of holding it.

Since the information of $f_{\max}$, $f_{\min}$ and t are updated each iteration and the number of parallel simplex processing is dynamic, a Master/Slave archi-

3.6 Parallel Implementation

Figure 3.6: The parallel computation model for TRIOPT.

structure is suitable for TRIOPT to carry out parallel computation.

Suppose there are totally p processors. As indicated in Figure 3.6, the master processor, P_0 , manages the working list $\mathcal{L}_w$, sends simplices to slave processors. The slave processes, $P_1, \dots, P_{p-1}$, receive simplices from the master processor and either split the simplices and send the child simplices back to the master processor, or do a Nelder-Mead local search when the local search option is activated in TRIOPT.

Algorithm 3.2 presents the pseudo code of the master processor of parallel implementation of TRIOPT without local search. In each assessment cycle, or iteration, the master processor distributes the simplices with entropy $E \geq \beta$ to slave processors and samples which need to be evaluated are assessed there. The working status of slave processors is monitored by the master processor. If the message tag from slave processor is IDLE, the master processor sends a simplex to it and sets its status to SIMP accordingly if there are simplices to be evaluated; otherwise, the status of the slave processor is set to IDLE. If the message tag is other than IDLE, then the master processor inserts the message content into the temporary list $\mathcal{L}_t$ and begins another

3.6 Parallel Implementation

Algorithm 3.2 Parallel TRIOPT—Master

```

1: initial partition;
2:
3: while termination criteria are not met do
4:   repeat
5:     RECV_MESSAGE ( $\mathcal{S}_i$ , any slave processor, RECVTAG);
6:     if RECVTAG is IDLE then
7:       set the slave processor status to IDLE
8:       if  $\exists E_{\mathcal{S}_c} \geq \beta$  in  $\mathcal{L}_w$  then
9:         SENDTAG = SIMP;
10:        select  $\mathcal{S}_c$  from  $\mathcal{L}_w$ ;
11:        SEND_MESSAGE ( $\mathcal{S}_c$ , the slave processor, SENDTAG);
12:        set the slave processor status to SIMP;
13:      end if
14:    else
15:      if  $H_{\mathcal{S}_i} < \delta$  then
16:        insert  $\mathcal{S}_i$  into  $\mathcal{L}_p$ ;
17:      else
18:        insert  $\mathcal{S}_i$  into  $\mathcal{L}_t$ ;
19:      end if
20:    end if
21:  until all slave processors are IDLE
22:  concatenate  $\mathcal{L}_t$  and  $\mathcal{L}_w$ ;
23:  flush  $\mathcal{L}_t$ ;
24:  update  $f_{\max}$ ,  $f_{\min}$ ,  $t$ ;
25: end while
26:
27: for each slave processor do
28:   SENDTAG = QUIT;
29:   SEND_MESSAGE ( $\emptyset$ , slave processor, SENDTAG);
30: end for

```

message receiving operation. The assessment cycle is finished when all slave processors are IDLE. Then list $\mathcal{L}_t$ and $\mathcal{L}_w$ are concatenated and $\mathcal{L}_t$ is flushed.

Algorithm 3.3 illustrates the work done by a slave processor. It is rather simple compared with the master processor since its main purpose is computation. First, the slave processor sends a message to master processor with an IDLE tag. This initiates the communication between slave and master

3.6 Parallel Implementation

Algorithm 3.3 Parallel TRIOPT—Slave

```

1: SENDTAG = IDLE;
2: SEND_MESSAGE ( $\emptyset$ , master, SENDTAG);
3:
4: loop
5:   RECV_MESSAGE ( $\mathcal{S}_c$ , master, RECVTAG);
6:   if RECVTAG is QUIT then
7:     quit;
8:   else
9:     calculate the location of new sample  $\mathbf{x}_{\text{new}}$ ;
10:    evaluate  $f(\mathbf{x}_{\text{new}})$ ;
11:    split  $\mathcal{S}_c$  into child simplices  $\mathcal{S}_i$ ;
12:    for each child simplex  $\mathcal{S}_i$  do
13:      SENDTAG = SIMP;
14:      SEND_MESSAGE ( $\mathcal{S}_i$ , master, SENDTAG);
15:    end for
16:  end if
17:  SENDTAG = IDLE;
18:  SEND_MESSAGE ( $\emptyset$ , master, SENDTAG);
19: end loop

```

processors. Then the slave processor enters an infinite loop which consists of a message receiving and several message sending operations. If the message tag from the master processor is QUIT, then the slave processor terminates; otherwise, it receives the simplex, splits it and sends the child simplices back to the master processor. After all child simplices are sent, the loop ends with a sending message operation with tag IDLE which indicates the slave processor is now ready to execute a new loop.

The local search option can be easily included by adding a list $\mathcal{L}_{nm}$ on the master processor to manage the candidates of local search and circulate them when some slave processors are idle. On the slave processors, the received simplex is used as the starting simplex of Nelder-Mead method when the tag is LOCAL. The main characteristic of parallel implementation of TRIOPT is that the number of parallel function evaluations is adaptive, which is an

advantage when the computation capacity of each slave processor is not the same. Those can do faster computations will evaluate more simplices than those are slower. In addition, incorporating the local search into the parallel computation model make it possible to execute local and global searches in parallel.

3.6.3 Parallel Performance

In each assessment iteration, TRIOPT splits those simplices with entropies $E \geq \beta$ which makes the number of function evaluations adaptive. Since parallel TRIOPT adopts a Master/Slave architecture, the total number of function evaluations will be the same as the serial model. That is, a linear speedup will be the ideal performance of parallel TRIOPT. When the objective functions are time-consuming to evaluate, the overhead of communication between processors can be ignored. The speedup can be decided by the parallel function evaluations in each assessment cycle.

As an example, Figures 3.7 and 3.8 show the number of parallel function evaluations taken by TRIOPT on Six-Hump problem during each assessment cycle with 1.0% accuracy and 0.01% accuracy, respectively. It can be observed that the maximum number of function evaluations is 12 in both accuracy levels, while the minimum is 2 in 1.0% accuracy level and 4 in 0.01% accuracy level. In both levels, the adaptiveness of TRIOPT is quite obvious.

Figure 3.9 depicts the speedup with different number of slave processors running the parallel implementation of TRIOPT for the Six-Hump problem with 1.0% and 0.01% accuracy levels. It can be noticed that when using 4 processors or less, the speedup is quite good, but the speedup with 8 processors is not very satisfactory, and furthermore, the speedup gets little

3.6 Parallel Implementation

Figure 3.7: Number of parallel function evaluations taken on Six-Hump with 1.0% accuracy.

Figure 3.8: Number of parallel function evaluations taken on Six-Hump with 0.01% accuracy.

improvement when moving from 8 processors to 16. Considering the number of function evaluations needed for the two levels of accuracy (53 for 1.0% and 82 for 0.01%), the speedup is acceptable since the total number of parallel evaluations is rather small at each accuracy level. In addition, the maximum number of concurrent function evaluations during each assessment cycle for both accuracy level is 12, which rationalizes the performance on 16 processors since not all processors are utilized even when the maximum number of parallel function evaluations occurs during execution. It can be expected

3.6 Parallel Implementation

Figure 3.9: Speedup on the Six-Hump problem.

Figure 3.10: Speedup on the Rastrigin(5) problem.

3.6 Parallel Implementation

Table 3.9: Number of parallel function evaluations.

<i>Problem</i>	<i>1.0%</i>			<i>0.01%</i>		
	<i>min.</i>	<i>avg.</i>	<i>max.</i>	<i>min.</i>	<i>avg.</i>	<i>max.</i>
Six-Hump	2	5.9	12	4	5.9	12
G & P	4	6.0	12	4	6.0	12
Ackley	3	7.5	16	1	7.5	16
Spike	1	4.2	14	1	4.5	14
Schwefel	1	5.3	16	1	5.4	24
Easom	3	4.2	5	3	4.2	5
Crescent	1	3.0	5	1	3.0	5
Griewank(2)	2	6.3	13	1	4.5	16
Griewank(3)	1	8.3	49	1	8.3	49
Griewank(4)	3	11.6	47	2	10.6	76
Rastrigin(2)	1	6.0	14	1	6.0	14
Rastrigin(3)	8	18.2	48	8	18.2	48
Rastrigin(4)	6	49.4	215	6	49.4	215
Rastrigin(5)	1	152.7	387	1	152.7	387
ANVCS	2	4.0	6	2	6.0	14
Truss structure	1	3.0	5	4	7.4	14
Weeds	3	15.8	52	3	15.8	52
Palmer 4	4	6.0	10	3	9.7	16
Pressure vessel	3	7.5	16	3	7.5	16

when the number of function evaluations and the maximum number of function evaluations during each assessment cycle go up, better performance on speedup can be achieved. Figure 3.10 exhibits the speedup on the Rastrigin(5) problem, from which an almost linear speedup can be observed.

The number of minimum, average and maximum number of function evaluations during each assessment cycle for all test problems except the Protein folding problems are summarized in Table 3.9. Table 3.10 presents such statistics for the Protein folding problems. The maximum number of function evaluations reaches 387 among all test problems, while the minimum number is only 1. The average number of samples evaluated during each assessment cycle ranges from 3.0 to 152.7. For the Protein folding problems,

Table 3.10: Number of parallel function evaluations for Protein folding problems

<i>Number of function evaluations</i>	<i>min.</i>	<i>avg.</i>	<i>max.</i>
Protein folding 100 evaluations	4	7.3	18
Protein folding 300 evaluations	4	10.1	28
Protein folding 500 evaluations	4	19.2	116
Protein folding 1000 evaluations	3	15.7	146
Protein folding 2000 evaluations	2	14.3	154

it can also be noticed that the adaptiveness of TRIOPT is quite impressive. For example, the maximum number of evaluations for 500 function evaluation level reaches 116, which is almost one fourth of the total evaluations allowed. The flexible number of concurrent simplices to be processed enables the parallel model adaptable to any number of parallel processors. It is also possible that the number of parallel processors changes in a dynamic manner according to currently available computation capacities during execution.

3.7 Summary

In this chapter, the motivation of using triangulated partitioning is explained; the function value transformation and simplex evaluation are presented; two splitting schemes are designed and their advantages and disadvantages are illustrated; and the algorithm TRIOPT is described and numerical experiments are conducted to show its performance. The adaptiveness of concurrency of TRIOPT makes it adopt easily to parallel implementation; Master/Slave architecture is utilized according to its algorithmic features.

Chapter 4

Constrained Optimization Using Interval Analysis

In this chapter, first a brief introduction to interval analysis is given; then three interval subdivision strategies are proposed for constrained optimization. The first one is based on the information gained from the change of the objective function range during the interval splitting; the second one bases on the constraint violation; and the last one utilizes the constraint infeasibility index. These rules are applied to constrained global optimization problems and numerical results are compared with a standard interval analysis approach. Parallel computation models with blocking and non-blocking message passing are provided; their characteristics and performance are analyzed.

4.1 Introduction

Interval analysis for optimization employs the concept of branch and bound. The whole search area is consecutively subdivided into small search areas—branching. Whereas the interval arithmetic provides the range of function values—bounding, which eliminates certain areas which surely do not hold

Algorithm 4.1 Interval analysis for constrained optimization

Require: list $\mathcal{L}$;**Ensure:** solution;

```

1:  $\mathcal{L} = \emptyset$ ;
2: set the whole search area as an initial box and insert it into  $\mathcal{L}$ ;
3: while  $\mathcal{L} \neq \emptyset$  do
4:   select a box  $\mathcal{B}$  from  $\mathcal{L}$ ;
5:   subdivide the box  $\mathcal{B}$  into subsets  $\mathcal{B}_i$  from selected direction;
6:   for each  $\mathcal{B}_i$  do
7:     evaluate objective and constraint functions over the box  $\mathcal{B}_i$ ;
8:     if better solution is found then
9:       update current best solution and  $\mathcal{L}$ ;
10:    end if
11:    if  $\mathcal{B}_i$  cannot be discarded then
12:      insert  $\mathcal{B}_i$  into  $\mathcal{L}$ ;
13:    end if
14:  end for
15: end while
16: report best solution

```

the optima. Algorithm 4.1 presents a prototype of interval analysis for constrained optimization. The whole search space is first treated as an interval (a box), which is subdivided into several sub-intervals. The procedure is then recursively applied to those sub-intervals and results in a tree of sub-intervals.

In Step 4 of Algorithm 4.1, the box selection step, several strategies can be applied. Such as, select the box with best upper bound (Skelboe, 1974); select the box which has been longest in the list (Hansen and Sengupta, 1980); and select the box which has maximum width (Hansen and Sengupta, 1980) and so on.

In Step 5 of Algorithm 4.1, the subdivision step, the direction can be selected according to a merit function:

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

where $\mathcal{J} = \{1, 2, \dots, n\}$ and $D(i)$ is determined by the given rule. Let X_i denote an interval over the variable x_i ; $w(\cdot)$ stand for the width of an interval; $m(\cdot)$ indicate the midpoint of an interval; capital letters F , G , H and C for the inclusion functions of f , g , h and c . Some subdivision strategies are listed as follows:

Subdivide from the Widest Edge This rule is called interval width oriented rule (Moore, 1966; Skelboe, 1974; Ratschek and Rokne, 1988), it chooses the coordinate direction with

$$D(i) = w(X_i).$$

The Rule of Hansen and Walster (Hansen, 1992), this rule selects the subdivision direction by using

$$D(i) = w(\nabla F_i(X)) \cdot w(X_i).$$

It is a heuristic direction selection rule which aims to find the component with the largest value of

$$W_i = \max_{t \in X_i} f(m(X_1), \dots, m(X_{i-1}), t, m(X_{i+1}), \dots, m(X_n)) \\ - \min_{t \in X_i} f(m(X_1), \dots, m(X_{i-1}), t, m(X_{i+1}), \dots, m(X_n)).$$

The factor W_i , which is assumed to reflect how much f varies as x_i varies over X_i , is then approximated by $w(\nabla F_i(X)) \cdot w(X_i)$.

The Rule of Ratz The rule of Ratz (Ratz, 1992; Ratz and Csendes, 1995)

can be formulated as

$$D(i) = w\left(\nabla F_i(X) \cdot (X_i - m(X_i))\right).$$

The underlying idea is to minimize the width of the following inclusion function:

$$\begin{aligned} w(F(X)) &= w\left(F(X) - f(m(X))\right) \\ &\approx w\left(\nabla F(X) \cdot (X - m(X))\right) \\ &= w\left(\sum_{i=1}^n \frac{\partial F}{\partial x_i}(X) \cdot (X_i - m(X_i))\right) \\ &= \sum_{i=1}^n w\left(\frac{\partial F}{\partial x_i}(X) \cdot (X_i - m(X_i))\right). \end{aligned}$$

Those rules have been widely applied in unconstrained global optimization. However, rules which utilize gradient information cannot be applied when such information is not available.

In constrained optimization, a box can be discarded if one or more conditions listed below are met:

- The box is proved to be infeasible;
- The upper bound of the box is less than a known feasible solution;
- The width of its objective function range is less than a certain tolerance ε_f . This rule is applied for computational reason since theoretically there always exists a gap between the upper and lower bounds no matter how small an interval is.

The feasibility of a box is checked by examining the range of each constraint. However, when dealing with equality constraints, it is almost com-

putationally impossible to satisfy them (Ratschek and Rokne, 1995). In computation, the equality constraints can be relaxed and replaced by:

$$-\varepsilon \leq h_i(x) \leq \varepsilon \quad i = r + 1, \dots, s, \quad (4.1)$$

for some positive small valued ε . Similarly, the inequality constraints can also be relaxed:

$$g_i(x) \leq \varepsilon \quad i = 1, \dots, r.$$

Thus, infeasible boxes can be discarded and feasible boxes can be kept for later processing if their upper bounds are greater than the known best feasible solution. Since the constraints are also evaluated by interval arithmetic, there exist certain boxes neither totally feasible nor completely infeasible. Those boxes are called “*pending boxes*”.

4.2 New Interval Subdivision Strategies

Interval subdivision strategies affect the performance of interval analysis a lot. In unconstrained optimization, certain accelerating devices, such as the monotonicity test, can discard a box when the objective function over the box is monotonic. But for constrained problems, the monotonicity test of the objective function over a box is not sufficient to discard the box, unless the box is totally feasible, which might be very difficult to achieve in constrained optimization, especially for highly constrained problems. Such disadvantages make the evaluation of gradient sometimes unnecessary or not very helpful, which consequently make the subdivision strategies based on

the gradient information not applicable to constrained optimization. Sometimes the objective function is simply non-differentiable, which also prevents the utilization of strategies based on gradient information.

4.2.1 Subdivision Based on Information Gain

During box splitting, the objective function range and the constraint functions ranges over a box will get tighter as the “size” of the box decreases. If the information of such ranges changes can be utilized, it might be helpful for the selection of subdivision direction, that is, split from certain direction will reduce the ranges quicker than from others, which consequently leads to a faster convergence.

Suppose currently the box shown in Equation (4.2) is being processed,

$$\begin{aligned} f(\mathbf{x}) &= (3x_1^2 + x_2^2) \exp(1 - x_1^2 - x_2^2), \\ (x_1 &= [-3.5, 3.5], x_2 = [0.0, 7.5]), \end{aligned} \tag{4.2}$$

with a known solution of $f(\mathbf{x}) = 1.0$. Assume the function range over the box can be calculated tightly.

A traditional approach is that the box is always split from the widest edge, as shown in Figure 4.1. That is, first the box is split from the x_2 direction depicted in Figure 4.1(a) since it has the widest edge. The two boxes after split are plotted in Figures 4.1(b) and 4.1(c). After the first splitting, the box in Figure 4.1(b) can be discarded because its upper bound is less than the known solution. The second split is carried out from the x_1 direction as shown in Figure 4.1(c) since now x_1 is the widest edge. The boxes plotted in Figures 4.1(d) and 4.1(e) are the results of the second splitting. No box can be discarded because the function ranges are the same as the one before

4.2 New Interval Subdivision Strategies

Figure 4.1: Split from the widest edge.

splitting. The results of two split operations (with four function evaluations) eliminate one half of the original search space.

The split shown in Figure 4.2 gives totally different split results. Suppose the first split is still carried out from the direction with the widest edge. After the first splitting, one half of the search space can be dropped and it suggests that split from x_2 is helpful. Suppose the next split operation continues select

4.2 New Interval Subdivision Strategies

Figure 4.2: Split by information gain.

the direction x_2 to split, and gives the boxes plotted in Figures 4.2(d) and 4.2(e). Again, the box in Figure 4.2(d) can be discarded. The results of two split operations (with four function evaluations) eliminate three fourths of the original search space.

The example shown in Figures 4.1 and 4.2 is only an explanatory one, but it does illustrate the fact that the information acquired during box splitting

4.2 New Interval Subdivision Strategies

provides some useful indication for subdivision direction selection.

Suppose a box $\mathcal{B}$ is split into m boxes, $\mathcal{B}_i$, $i = 1, \dots, m$, the information gain is calculated by comparing the objective function ranges over the child boxes to the objective function range over the parent box.

For all feasible and pending child boxes, the information gain, I_G , is calculated by

$$I_G = \frac{1}{k} \sum_{i=1}^k (1 - r_i \exp(1 - r_i)), \quad (4.3)$$

where,

$$r_i = \frac{\overline{F}(\mathcal{B}_i) - \underline{F}(\mathcal{B})}{\overline{F}(\mathcal{B}) - \underline{F}(\mathcal{B})},$$

$\overline{F}$ and $\underline{F}$ are upper and lower bounds of the objective function, k is the number of feasible and pending boxes, for unconstrained optimization, $k = m$. If a child box is infeasible, the box can be discarded immediately without assigning information gain to it. The information gain reflects the change of child boxes over parent box, the bigger the change is, the higher the information gain is. Therefore, if a split from one direction results in a big shrink of function range, the information gain will indicate that splitting from that direction is helpful. The formula is adopted from Equation 3.2, but the contribution of r_i is the inverse of contribution of $\ddot{f}(\mathbf{x})$ in Equation 3.2.

To use the strategy of information gain, each direction should be split at least once. Then the direction which gives maximum information gain can be selected for subsequent subdivision. However, if one direction always has the highest information gain, the procedure will select this direction consecutively and result in very irregular boxes. To avoid this, an edge width tolerance,

4.2 New Interval Subdivision Strategies

Algorithm 4.2 Direction selection based on information gain**Require:** the box $\mathcal{B}$, the width tolerance ε_w , temporary variable u ;**Ensure:** the direction d to split the box $\mathcal{B}$;

```

1:  $u = \max w(X_i)$ ,  $d = \arg \max w(X_i)$ ;
2: if all directions have been split at least once then
3:   for  $i = 1$  to  $n$  do
4:     if  $u < I_{G_i}$  and  $w(X_i) > \varepsilon_w$  then
5:        $u = I_{G_i}$ ;
6:        $d = i$ ;
7:     end if
8:   end for
9: else
10:  select the first direction from those have not been split as  $d$ ;
11: end if

```

ε_w , is set to prohibit the procedure to select a direction for subdivision when its width is less than ε_w . The procedure of subdivision utilizing information gain is shown in Algorithm 4.2.

Since the information gain approach only utilizes information of the objective function, it can be applied to both unconstrained and constrained optimization. To show the impact of information gain I_G on subdivision direction selection, it is tested on the set of theoretical problems in Chapter 3 and compared to the approach of splitting from the widest edge. Testing on unconstrained problems avoids the influence of constraints and the effectiveness of information gain can be better demonstrated. Both procedures always select the box with maximum upper bound from the list $\mathcal{L}$, the only difference is the split direction. The approach to select the maximum upper bound and split from the widest edge is rather a standard procedure in interval analysis and usually used as a benchmark for other methods. The middle point is evaluated to speedup convergence.

Table 4.1 presents the results of widest edge and information gain approaches on unconstrained problems, where “*eva.*” stands for the number

4.2 New Interval Subdivision Strategies

Table 4.1: Performance of information gain on unconstrained problems.

Problem	widest edge		information gain			
	eva.	len.	eva.	% eva.	len.	% len.
Six-Hump	9784	776	5791	59.12	506	65.21
G & P	589438	36102	565819	96.00	36269	100.46
Ackley	676	104	625	92.45	87	83.65
Spike	766	71	757	98.83	72	101.01
Schwefel	46282	5452	46207	99.84	5431	99.61
Easom	64	8	64	100.00	8	100.00
Crescent	10606	528	3658	34.49	100	18.94
Griewank(2)	103	15	103	100.00	15	100.00
Griewank(3)	157	24	157	100.00	24	100.00
Griewank(4)	211	34	211	100.00	34	100.00
Rastrigin(2)	118	16	118	100.00	16	100.00
Rastrigin(3)	172	23	172	100.00	23	100.00
Rastrigin(4)	226	30	226	100.00	30	100.00
Rastrigin(5)	280	37	280	100.00	37	100.00
Average				91.48		90.63

of function evaluations (that is, interval evaluations); “len.” indicates the maximum length of $\mathcal{L}$ when solving the problem. The two columns with “%” under “information gain” is the percentage of function evaluations and maximum length compared with the widest edge approach. It can be observed that for Easom, variants of Griewank and Rastrigin, the performance of both approaches is same, the information gain approach does not help too much on these problems. But these problems seem to be easily solved by interval analysis since the number of function evaluations is rather small for each problem. G & P, Ackley, Spike and Schwefel get slight improvements range from 0.16% to 7.55%. Two remarkable improvements are made by Six-Hump and Crescent. Six-Hump takes 59.12% and Crescent only costs 34.49% function evaluations used by the widest edge approach. It should also be pointed out that Crescent is a non-differentiable function. The percentage of maxi-

4.2 New Interval Subdivision Strategies

mum list length gives a similar trend as the number of function evaluations. The average percentage is given in the last row in Table 4.1, the information gain approach costs 91.48% function evaluations compared with widest edge approach, and reduces the complexity to manage the list to 90.63%. In Csendes and Ratz (1997), different subdivision direction strategies are studied. Compared with the results provided there, the improvement achieved by information gain is as competitive as those utilizing gradient information.

4.2.2 Subdivision Based on Constraint Violation

The information gained during box splitting proved to be helpful for subdivision direction selection, but it only utilizes the information of the objective function. In constrained optimization, since all constraints should be satisfied, thus, a feasible solution can be quickly found if the infeasible parts of constraints can be reduced effectively. Define *constraint violation* as:

Definition 4.1. The **constraint violation**, V_i , of a certain constraint c_i is the part that c_i does not satisfy the constraint.

Therefore, each box in the list $\mathcal{L}$ will have a constraint violation vector, $V^T = (V_1, V_2, \dots, V_s)$, where s is the number of constraints. If a constraint c_i is satisfied, then obviously the constraint violation $V_i = 0$; if the constraint is pending, the constraint violation V_i is:

$$V_i = \begin{cases} \overline{C}_i & \text{if the constraint is a } c_i \leq 0 \text{ type;} \\ \overline{C}_i - \underline{C}_i & \text{if the constraint is a } c_i = 0 \text{ type,} \end{cases} \quad (4.4)$$

where $\overline{C}_i$ and $\underline{C}_i$ are the upper and lower bounds of constraint c_i , respectively.

The constraint-variable relationship matrix, R , which is an $n \times s$ matrix

4.2 New Interval Subdivision Strategies

Algorithm 4.3 Direction selection based on constraint violation**Require:** the box $\mathcal{B}$, the width tolerance ε_w , temporary variable u , q ;**Ensure:** the direction d to split the box $\mathcal{B}$;

```

1: if  $\mathcal{B}$  is feasible then
2:    $d = \arg \max w(X_i)$ ;
3: else
4:    $q = \arg \max V_i$ ;
5:    $d = \arg \max w(X_i|c_q)$ ,  $u = \max w(X_i|c_q)$ ;
6:   if  $u < \varepsilon_w$  then
7:      $d = \arg \max w(X_i)$ ;
8:   end if
9: end if

```

where each column corresponds to a constraint, is constructed to show the contribution of each variable to each constraint:

$$R = \begin{bmatrix} 1 & 0 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 1 & \cdots & 1 \end{bmatrix}. \quad (4.5)$$

That is, if a variable x_i contributes to a constraint c_j , then the corresponding element $R_{i,j}$ is set to 1; otherwise, the element is set to 0.

Algorithm 4.3 presents the procedure to select subdivision direction based on constraint violation. If the box is feasible, then the edge with maximum width is selected; otherwise, the constraint c_q with maximum constraint violation is first identified, then the variables related to constraint c_q are distinguished via the constraint-variable relationship matrix R , among them, the one with maximum width is assigned as the subdivision direction. Similar to the information gain approach, a width tolerance ε_w is applied. When the identified direction has a width less than ε_w , the edge with maximum width is assigned for subdivision instead.

By splitting a direction related to the most violating constraint, it is

expected that:

1. the constraint violation of that constraint can be reduced since the variable (direction) selected certainly contributes to the constraint; consequently, the chance of finding a feasible solution increases;
2. the subdivided boxes can be discarded at an early stage since the constraint violation is high.

4.2.3 Subdivision Based on Infeasibility Index

The approach of infeasibility index, which is similar to the constraint violation strategy, is also based on the information of constraints.

Definition 4.2. *The infeasibility index, I_{F_i} , of a constraint c_i is the ratio of its infeasible part to its feasible part.*

That is,

$$I_{F_i} = \frac{\text{Infeasible part of } c_i}{\text{Feasible part of } c_i}.$$

Or, more specifically,

$$I_{F_i} = \begin{cases} |\overline{C}_i|/|\underline{C}_i| & \text{if the constraint is a } c_i \leq 0 \text{ type;} \\ |\overline{C}_i - \underline{C}_i|/2\varepsilon & \text{if the constraint is a } c_i = 0 \text{ type.} \end{cases} \quad (4.6)$$

Where $\overline{C}_i$ and $\underline{C}_i$ again are the upper and lower bounds of constraint c_i , respectively. ε is the constraint relaxation value used in Equation (4.1).

The procedure to select the subdivision direction based on infeasibility index is similar to Algorithm 4.3, the difference is in Step 4: instead of

selecting the direction with maximum constraint violation, the constraint with maximum infeasibility index is identified.

4.3 The Algorithm

The algorithms for constrained optimization in this research are constructed based on Algorithm 4.1. In Step 4 of Algorithm 4.1, the box with maximum upper bound is selected for processing, that is, best-first strategy for box selection. In Step 5, the subdivision is carried out utilizing the three subdivision strategies introduced in Section 4.2. Since the basic framework of the three approaches are same, they are combined to be called COPIA (Constrained OPTimization using Interval Analysis). Each subdivision strategy then corresponds to a variant of COPIA.

COPIA follows a branch and bound approach, which is an intelligent search heuristic for finding global optima solutions to various problems. Basic branch and bound searches the feasible region by iteratively subdividing the feasible region and recursively searching each piece for an optimal feasible solution, that is, it is a search-based enumeration technique that enumerates the entire solution space implicitly. However, it is often more efficient because it can eliminate regions that do not contain an optimal solution.

Convergence properties for branch and bound algorithms were addressed by several earlier works. In particular, Horst and Tuy (1996) investigate a general branch and bound frame for global optimization, which includes problems in the form of Equation (1.1). More explicit convergence properties of interval branch and bound approaches are given by, such as, Ratschek and Rokne (1988), Ratschek and Rokne (1995) and Csendes (2001) and so on. The convergence property of COPIA can be proved in a similar way.

4.4 Numerical Experiments

4.4.1 Test Problems

The test bed for COPIA consists of two batches of problems. The first batch includes Problems 1 to 10. The dimension of the problems ranges from 2 to 6. They are used to examine the performance of the three variants of COPIA. In the parallel implementation of COPIA, they are also tested with 1, 2, 4, 8, 16 processors, respectively, to inspect the performance of parallel implementation of COPIA using blocking message passing. The second batch contains Problems 11 to 16. The maximum dimension of second batch problems is up to 20. They are tested together with the first batch problems with 1, 2, 4, 8 and 16 processors using the non-blocking message passing parallel computation model of COPIA. The problems are solved with the best variant of COPIA observed from the numerical results in the first batch problems. Table 4.2 lists the summary of the constrained test problems with their dimensions, number and type of constraints and references. Detailed description of the test problems can be found in Appendix B.

4.4.2 Implementation Details

The three variants of COPIA are tested for the first batch of test problems in the numerical studies with a serial implementation. As a benchmark, the approach which always selects the widest edge for subdivision is also incorporated. All four approaches select the box with maximum upper bound for processing, that is, best-first strategy for box selection. A box is always split into two child boxes from the direction selected, that is, bisection for box splitting. The midpoint test is applied for quick convergence. To sum

4.4 Numerical Experiments

Table 4.2: Summary of constrained test problems.

<i>Problem</i>	<i>Dim.</i>	<i>Constraints</i>	<i>Reference</i>
1	5	3 equality	Floudas et al. (1999)
2	6	4 equality, 1 inequality	Floudas et al. (1999)
3	5	6 inequality	Floudas et al. (1999)
4	2	3 inequality	Floudas et al. (1999)
5	2	3 inequality	Spike + 3 constraints
6	5	2 equality, 3 inequality	Floudas et al. (1999)
7	3	3 inequality	Floudas et al. (1999)
8	5	6 inequality	Floudas et al. (1999)
9	4	1 equality, 1 inequality	Rastrigin(4) + 2 constraints
10	2	3 inequality	Crescent + 3 constraints
11	8	6 inequality	Floudas et al. (1999)
12	6	4 equality, 1 inequality	Floudas et al. (1999)
13	6	6 inequality	Floudas et al. (1999)
14	8	1 equality, 1 inequality	Rastrigin(8) + 2 constraints
15	10	1 equality, 1 inequality	Rastrigin(10) + 2 constraints
16	20	14 equality	CUTEr ^a

^a CUTEr: A Constrained and Unconstrained Testing Environment, revisited. <http://cutter.rl.ac.uk/cute-www/problems.html>.

up, the approaches used for numerical studies are:

widest edge: split from the direction with the widest edge;

information gain: split from the direction which gives the biggest information gain. Algorithm 4.2 illustrates how the direction is selected;

constraint violation: split from the direction which gives the largest constraint violation. Algorithm 4.3 is used to identify which direction should be picked; and

infeasibility index: split from the direction which gives the highest infeasibility index.

The parameters of different tolerance (edge tolerance ε_w , objective function tolerance ε_f and constraint tolerance ε) are: for Problems 1, 2, 3, 8 and

9, $\varepsilon_w = \varepsilon_f = \varepsilon = 10^{-3}$, for Problems 4, 5, 6, 7 and 10, $\varepsilon_w = \varepsilon_f = \varepsilon = 10^{-4}$.

4.4.3 Numerical Results

Table 4.3 presents the results for the first batch test problems using the four subdivision strategies. Where “*fun. eva.*” indicates the number of objective function evaluations, “*con. eva.*” stands for constraint evaluations, and “*length*” gives the maximum length of $\mathcal{L}$ when solving the problem. The last row with “Avg” displays the average percentage of all problems of “*fun. eva.*”, “*con. eva.*” and “*length*” each approach takes compared with the widest edge approach.

From Table 4.3, it can be observed that the information gain approach uses less objective function and constraint evaluations than the widest edge approach for all problems except Problem 2, which costs a little bit more objective function and constraint evaluations. The maximum length of working list $\mathcal{L}$ gives slight different results: all problems take shorter list length except Problems 4 and 5 which need the same list length as the widest edge approach does. Note the list length of Problem 10 is quite short. The overall performance of information gain compared with the widest edge approach is shown in the last row in Table 4.3. The average percentage of “*fun. eva.*”, “*con. eva.*” and “*length*” are 86.76%, 88.98% and 85.19%, respectively.

The results of the constraint violation approach shown in Table 4.3 provide much better performance than the widest edge approach does. It outperforms the widest edge approach in every problem except Problem 7 in objective function evaluations and maximum list length. The performance of Problems 1, 2, 3, 6 and 8 is quite impressive. The overall results show that the constraint violation approach uses 74.61% objective function evaluations,

4.4 Numerical Experiments

Table 4.3: Performance of different subdivision selection strategies on the first batch test problems.

Prob.	widest edge			information gain			constraint violation			infeasibility index		
	fun.eva.	con.eva.	length	fun.eva.	con.eva.	length	fun.eva.	con.eva.	length	fun.eva.	con.eva.	length
1	235549	1139727	22857	215427	1047039	21084	169554	815690	13244	169554	815690	13244
2	873619	6130385	74026	874988	6141281	73992	181823	1394046	11486	59372	437082	3689
3	36288	123214	5118	33126	113277	4695	25293	88455	3489	23637	81993	3265
4	148	569	9	148	569	9	146	566	9	146	566	9
5	263	365	26	263	365	26	263	365	26	263	365	23
6	60567	214986	12403	58120	212106	11977	16229	60503	3964	13752	50830	3157
7	4879	425	746	4687	278	675	5136	350	764	3120	51	474
8	136519	498436	17126	127098	469103	16298	74239	274353	10378	80335	295577	9832
9	54138	96483	7941	34022	63651	5289	54138	96483	7941	54138	96483	7941
10	10030	653	511	3691	533	98	9878	216	505	9875	196	505
Avg				86.76%	88.98%	85.19%	74.61%	66.42%	73.54%	68.65%	56.97%	66.04%

4.4 Numerical Experiments

Figure 4.3: Ratio of function evaluations taken by different strategies compared with the widest edge approach.

66.42% constraint evaluations and 73.54% maximum list length of which the widest edge approach spends.

The performance of the infeasibility index approach shows the same tendency as the constraint violation approach does. All problems outperform the widest edge approach. Moreover, the performance is also quite good compared with the constraint violation approach except for Problem 8. Problems 2, 3, 6 and 7 provide notable improvement even when using the constraint violation approach as the benchmark. The overall results show that the infeasibility index approach uses 68.65% objective function evaluations, 56.97% constraint evaluations and 66.04% maximum list length compared with the widest edge approach.

Figures 4.3 to 4.5 present the ratios of objective function evaluations, constraint evaluations and maximum list length the three variants of COPIA take compared with the widest edge approach. It can be easily observed that the ratios are less than 1.0 in most cases. The best one even only uses

4.4 Numerical Experiments

Figure 4.4: Ratio of constraint evaluations taken by different strategies compared with the widest edge approach.

Figure 4.5: Ratio of maximum list length taken by different strategies compared with the widest edge approach.

less than 10% of objective function evaluations, constraint evaluations and maximum list length of which the widest edge approach takes.

From the test results of information gain on unconstrained and constrained problems shown in Tables 4.1 and 4.3, it can be concluded that the information gain approach can provide better results than simply subdivide from the widest edge. The two variants of COPIA utilizing the constraint information, namely, the constraint violation approach and infeasibility index approach, also prove to be successful compared with the widest edge approach. This indicates improvement can be achieved when the information of objective function and constraints is properly handled in constrained optimization. The advantage of these subdivision rules lays on the fact that they do not need any information about the derivatives, which can not only save computational time but also be advantageous over other derivative-dependent approaches when such information cannot be acquired. Furthermore, the performance improvement is achieved based on currently available information, no additional information is needed; moreover, the extra computation effort needed is trivial.

4.5 Parallel Implementation

Parallel implementation of COPIA is practical since each branch of the problem can be processed independently. In this section, first the topology of the parallel computation model is introduced; consequently, the models using blocking and non-blocking message passing are presented and their performance is investigated.

Figure 4.6: The ring topology of the parallel model.

4.5.1 Parallel Computation Model Topology

The parallel computation model employs a ring connected topology, that is, each processor has two neighbors: one on the left, the other on the right. The virtual connection of all processors of the model is depicted in Figure 4.6. Suppose there are totally p processors, and each processor is assigned a *rank* ranging from 0 to $p - 1$. The rank of left neighbor of processor P_i can be represented as $(i - 1 + p) \bmod p$, and the rank of right neighbor of processor P_i can be denoted as $(i + 1) \bmod p$. This simple calculation “connects” each processor with its left and right neighbors.

Messages are always sent to processors’ right neighbors, and are received from left neighbors. The content of a message depends on the current working status of the message sending processor. If the working list $\mathcal{L}$ is not empty, then the message tag is set to BOX and a box is selected, packed and sent to the right neighbor; otherwise, the message tag is set to IDLE and a message with empty content is sent to the right neighbor. Sending an empty message with an IDLE tag is just to keep the message flow continuous.

4.5.2 Blocking Message Passing Model

In the blocking message passing model, all processors are divided into two groups: *odd group* and *even group*, according to their ranks. Processors in the even group always first send messages to their right neighbors and then

4.5 Parallel Implementation

Figure 4.7: A typical parallel computation cycle

receive messages from their left neighbors. The processors in the odd group, on the other hand, will first receive messages from their left neighbors and then send messages to their right neighbors. Thus, when a processor in the even group is sending a message, its counterpart in the odd group is ready to receive the message, and vice versa. By performing this one-to-one matched sending-receiving operation, the communication time can be minimized since blocking message passing operations can only return after they are finished.

The parallel implementation of COPIA is denoted with an initial “P” stands for “parallel” as “*P-COPIA*”. For the ease of description, the concept of *computation cycle*, or *iteration* is defined as:

Definition 4.3. A **computation cycle**, or **iteration** in *P-COPIA* is the time cycle from one message passing operation to the next same message passing operation.

A computation cycle in *P-COPIA* can be divided into three phases: sending-receiving phase, processing phase and gathering phase. Figure 4.7 plots a typical computation cycle. In the sending-receiving phase, the mes-

4.5 Parallel Implementation

sages are sent and received according to the above mentioned mechanism. The processing phase includes box splitting, interval evaluations, feasibility checking and working list management. At least one box is split when the working list $\mathcal{L}$ is not empty in each processing phase. Of course, it is also possible that more than one box are processed, the choice depends on the computation cost versus communication cost. The gathering phase, which is triggered when a better solution is found or a processor's working list $\mathcal{L}$ is empty, collects known best solutions from each processor and the working status of all processors to decide whether P-COPIA satisfies the termination criterion. Theoretically, a computation cycle in each processor will take almost the same amount of time since the sending and receiving operations are dependent on each other and the processing phases cost almost the same amount of time due to the characteristics of branch and bound.

In the processing phase, a box is selected for next subdivision according to predetermined rules. Here the concept *dominant* is used to describe such a box:

Definition 4.4. *A local dominant box is the box which can be selected according to the selection rules among all boxes on the processor it locates.*

Definition 4.5. *A global dominant box is the box which can be selected according to the selection rules among all boxes on all processors.*

In serial computation, a local dominant box, which at the same time is also a global dominant box, is selected for next subdivision. On a particular processor, P-COPIA works like a serial implementation if the message passing operations are neglected. The most important difference of P-COPIA from a serial model is that in the sending-receiving phase each processor first sends its local dominant box to its right neighbor. This difference makes the

Algorithm 4.4 P-COPIA using blocking message passing

```

1: repeat
2:   if  $\mathcal{L} \neq \emptyset$  then
3:     select a box  $\mathcal{B}_s$  from  $\mathcal{L}$  for sending;
4:     SENDTAG = BOX;
5:   else
6:      $\mathcal{B}_s = \emptyset$ ;
7:     SENDTAG = IDLE;
8:   end if
9:   if processor rank is even then
10:    SEND_MESSAGE ( $\mathcal{B}_s$ , left, SENDTAG);
11:    RECV_MESSAGE ( $\mathcal{B}_r$ , right, RECVTAG);
12:   else
13:    RECV_MESSAGE ( $\mathcal{B}_r$ , right, RECVTAG);
14:    SEND_MESSAGE ( $\mathcal{B}_s$ , left, SENDTAG);
15:   end if
16:
17:   if RECVTAG is BOX then
18:     insert  $\mathcal{B}_r$  into  $\mathcal{L}$  in descending upper bound order;
19:   end if
20:
21:   if  $\mathcal{L} \neq \emptyset$  then
22:     select a box  $\mathcal{B}$  from list  $\mathcal{L}$  for processing;
23:     subdivide  $\mathcal{B}$  into subsets  $\mathcal{B}_i$ ;
24:     for each  $\mathcal{B}_i$  do
25:       evaluate  $f(\mathcal{B}_i)$  and  $\mathbf{c}(\mathcal{B}_i)$ ;
26:       if better solution is found then
27:         update current best solution;
28:       end if
29:       if  $\mathcal{B}_i$  cannot be discarded then
30:         insert  $\mathcal{B}_i$  into  $\mathcal{L}$  in descending upper bound order;
31:       end if
32:     end for
33:   end if
34:
35:   ALLGATHER(current best solution)
36:   if current best solution is updated then
37:     update the working list  $\mathcal{L}$ ;
38:   end if
39:   ALLGATHER(processor status)
40: until all processors are idle

```

4.5 Parallel Implementation

dissemination of globally dominant boxes rather easily achievable. After a processor receives a box from its left neighbor, the local dominance of the box is checked. If it is locally dominant, the box is processed; otherwise, the box is inserted to the working list $\mathcal{L}$ and another local dominant box in $\mathcal{L}$ on this processor is selected and processed. Algorithm 4.4 presents a detailed description of a complete computation cycle with blocking message passing.

The characteristics of P-COPIA utilizing p processors can be summarized in the following lemmas and theorem.

Lemma 4.1. *In P-COPIA, if a box is globally dominant and its child boxes are locally dominant on all processors, then boxes split from this globally dominant box will pervade all processors after $p - 1$ computation cycles.*

Proof. Without loss of generality, suppose currently the processor P_0 has such a globally dominant box, $\mathcal{B}$. Since $\mathcal{B}$ is globally dominant, it is of course locally dominant on P_0 and is selected for subdivision. Consequently, one child box of $\mathcal{B}$ should be sent to P_1 in the next computation cycle since it is locally dominant on P_0 . That is, after 1st iteration, processor P_1 has a box split from $\mathcal{B}$. Similarly, P_2 will get such a box from P_1 after the 2nd iteration. Eventually, after $p - 1$ computation cycles, the processor P_{p-1} will acquire a box split from $\mathcal{B}$ once the local dominant properties of the child boxes hold. $\square$

Lemma 4.1 shows that the children of a globally dominant box will be processed by all processors if they cannot be discarded and locally dominant. This ensures the order of boxes to be processed is similar to the case in serial implementation.

4.5 Parallel Implementation

Lemma 4.2. *In P-COPIA, a processor can only be idle for at most $p - 1$ computation cycles consecutively if there is at least one box left among all processors.*

Proof. Suppose in an extreme case that currently only P_0 is processing a box, $\mathcal{B}$, all the remaining processors are idle. Also suppose that each subdivision will only result in one box, the other child boxes can be discarded. That is, no matter which processor is working, there only exists one box among all processors. Since in each computation cycle, a processor sends a message before processing a box, P_0 will send the child box split from $\mathcal{B}$ to P_1 after 1st iteration. Similarly, P_2 will get the chance to handle the only box left from P_1 after the 2nd iteration. Ultimately, the processor P_{p-1} will get a box after $p - 1$ iterations, and the idle time is exactly $p - 1$ computations cycles, which is the longest consecutive idle time among all processors. $\square$

Lemma 4.2 indicates the maximum idle computation cycles a processor might have consecutively, which can be used for termination detection. It also illustrates how the work load is balanced in an extreme case where only one box is left.

Lemma 4.3. *If all processors are not idle, the computation cycles on all processors will take almost the same amount of time.*

Proof. In each computation cycle, the time spend on sending-receiving phase of each processor is same due to the send-receive and receive-send operations in even and odd groups. The processing phase costs almost the same amount of time since each processor manipulates a homogeneous problem due to the characteristics of branch and bound algorithms. $\square$

4.5 Parallel Implementation

Based on Lemmas 4.1 to 4.3, the following theorem immediately comes into existence:

Theorem 4.4. *Each processor in P-COPIA will have almost the same work load, and the same data managing complexity.*

Here, the data managing complexity concerns the maximum list length of working list $\mathcal{L}$.

For parallel computation with p processors, the ideal linear speedup will be $S(p) = p$. However, due to the communication overhead, it is difficult to achieve such an ideal speedup. Suppose the number of computation cycles for a serial computation model is N_c , then for a good parallel computation model with p processors, each processor will need to process N_c/p computation cycles, which will make the total number of computation cycles the same as the serial model and the time spent by each processor the same. If denoting the time used for different phases of a computation cycle as T_{sr} (sending-receiving), T_c (processing) and T_g (gathering), the theoretical speedup $S(p)$ of parallel computation model with p processors can be described as:

$$\begin{aligned} S(p) &= \frac{N_c T_c}{(N_c/p)(T_{rc} + T_c + T_g)} \\ &= p \cdot \frac{T_c}{T_{rc} + T_c + T_g}. \end{aligned} \tag{4.7}$$

It can be observed that when the time spent on processing phase T_c becomes longer, the speedup $S(p)$ will become closer to p , which is the ideal case for a linear speedup. Of course, it is also possible that the parallel model explores the search space in a different way which results in a sub-linear speedup or a super-linear speedup.

4.5.3 Blocking Message Passing Performance

The parallel computation model with blocking message passing is tested in a cluster with up to 16 processors (Intel® Xeon™ 3.06 GHz). The first batch of problems is used for numerical studies. In the test problems, the processing phase in a computation cycle is relatively short compared with the sending-receiving phase and gathering phase. In real applications, the processing phase could be very time-consuming. The model with blocking message passing is designed for this kind of problems. To show the performance of P-COPIA on “costly” applications, a hypothetical cost of 50 milliseconds time is added to each function evaluation in Problems 1 to 10. Tests are carried out with 1, 2, 4, 8 and 16 processors, respectively. Since the basic framework of all variants of COPIA is the same, only the constraint violation variant is used for investigating the performance of the parallel model.

Table 4.4 provides the number of objective function evaluations, constraint evaluations and the maximum list length using different number of processors to solve Problems 1 to 10. It can be observed that the numbers of objective function and constraint evaluations using parallel model (2, 4, 8 and 16 processors) are very close to the serial model (1 processor); sometimes even less than the serial model. This indicates that P-COPIA using blocking message passing does not increase the effort needed to solve the problems.

The maximum list length, on the other hand, illustrates the work load distribution among all processors. Theoretically, it can be expected that as the number of parallel processors p increases, the maximum list length on each processor will decrease. The data in Table 4.4 verify this assumption. The maximum list length almost decreases correspondingly when the number of parallel processors increases for all problems except Problems 4 and

4.5 Parallel Implementation

Table 4.4: Objective function evaluations, constraint evaluations and maximum list length used with different number of processors.

<i>Problem</i>	<i>1</i>	<i>2</i>	<i>4</i>	<i>8</i>	<i>16</i>
<i>Objective function evaluation</i>					
1	169554	169558	169455	168736	169216
2	181823	181886	181582	178380	181498
3	25293	25235	25159	25257	25539
4	146	148	159	160	160
5	263	265	263	277	277
6	16229	16224	16280	15942	14951
7	5637	4686	6005	3672	2522
8	74239	74152	74921	75258	74641
9	54138	54116	54118	52982	51085
10	9878	9877	9783	9766	9667
<i>Constraint evaluation</i>					
1	815690	815688	815034	811871	814157
2	1394046	1393715	1392097	1370696	1391786
3	88455	88257	88107	88453	89315
4	566	580	615	619	619
5	365	365	365	380	380
6	60503	60510	60685	59802	56881
7	401	371	353	404	362
8	274353	274121	276767	278333	275682
9	96483	96536	96523	94579	91436
10	216	222	227	233	233
<i>Maximum list length</i>					
1	13244	6631	3998	1881	1120
2	11486	5937	3264	1685	832
3	3489	1768	941	518	272
4	9	6	4	4	4
5	26	14	10	6	6
6	3964	2001	1030	510	275
7	850	344	239	73	28
8	10378	5656	3034	1476	759
9	7941	3997	2054	997	568
10	505	268	153	100	60

5, which need rather short maximum list length and take relatively small number of objective function and constraint evaluations.

Figures 4.8 to 4.17 plot the ratios of objective function evaluations, constraint evaluations and the maximum list length of the parallel implementations compared with the serial model. An ideal linear speedup should be the case where the ratios of function evaluations and constraint evaluations equal to 1, and the maximum list length is $1/p$ of the serial model. It can be noticed that Problems 1, 2, 3 and 8 perform almost as good as in the ideal linear speedup case. Problems 4, 5 and 10 take a little more objective function or constraint evaluations (ratio slight greater than 1) than in the serial model. However, as discussed before, the numbers of evaluations are rather small when this happens. Notice that in Problem 10, where the number of objective function evaluations is near 10,000 while the number of constraint evaluations is around 220, only the ratio of constraint evaluations is greater than 1; the ratio of objective function evaluations, is slightly less than 1. The performance of Problem 7 is interesting, the number of objective function evaluations with 4 processors is bigger than the serial model while executions with 2, 8 and 16 exhibit super-linear speedups in terms of objective function evaluations. This indicates that the parallel model explores the search domain in a different way from the serial model since the order of boxes to be processed is slightly changed and Problem 7 is sensitive to such changes. Problems 6 and 9 present results even better than ideal linear speedup. The lines of maximum list length overlap with the ideal list length line, and the ratios of objective function and constraint evaluations gradually reduce as the number of processors increases.

Table 4.5 presents the time (in seconds) used with different number of

4.5 Parallel Implementation

Figure 4.8: Ratios of objective function evaluation, constraint evaluation and maximum list length for Problem 1.

Figure 4.9: Ratios of objective function evaluation, constraint evaluation and maximum list length for Problem 2.

4.5 Parallel Implementation

Figure 4.10: Ratios of objective function evaluation, constraint evaluation and maximum list length for Problem 3.

Figure 4.11: Ratios of objective function evaluation, constraint evaluation and maximum list length for Problem 4.

4.5 Parallel Implementation

Figure 4.12: Ratios of objective function evaluation, constraint evaluation and maximum list length for Problem 5.

Figure 4.13: Ratios of objective function evaluation, constraint evaluation and maximum list length for Problem 6.

4.5 Parallel Implementation

Figure 4.14: Ratios of objective function evaluation, constraint evaluation and maximum list length for Problem 7.

Figure 4.15: Ratios of objective function evaluation, constraint evaluation and maximum list length for Problem 8.

4.5 Parallel Implementation

Figure 4.16: Ratios of objective function evaluation, constraint evaluation and maximum list length for Problem 9.

Figure 4.17: Ratios of objective function evaluation, constraint evaluation and maximum list length for Problem 10.

4.5 Parallel Implementation

Table 4.5: Time used with different number of processors (in seconds).

<i>Problem</i>	<i>1</i>	<i>2</i>	<i>4</i>	<i>8</i>	<i>16</i>
1	7702.3	3870.5	1948.9	975.4	519.6
2	9795.6	4792.8	2279.0	1130.2	569.8
3	1175.2	586.0	297.4	150.5	79.6
4	7.6	4.3	3.4	3.4	3.7
5	7.9	4.3	2.9	2.8	3.3
6	635.7	318.8	160.7	81.6	44.4
7	62.6	30.5	17.4	7.9	4.1
8	3461.1	1729.0	875.8	441.1	218.1
9	1930.0	966.2	489.0	241.2	120.0
10	198.5	99.6	50.5	26.7	18.3

processors for Problems 1 to 10. The graphical displays of speedup versus processor number are depicted in Figures 4.18 to 4.21. It can be observed that P-COPIA exhibits an almost linear speedup in Figure 4.18 on Problems 1, 7 and 8. The real speedup lines are very close to the theoretical speedup line, which indicate P-COPIA has a good load balancing mechanism in these three problems. For Problems 3, 6 and 10 shown in Figure 4.19, the speedup degenerates when moving from 8 processors to 16 processors, but the performance is still acceptable. The speedups on Problems 4 and 5 plotted in Figure 4.20 give examples of weak performed parallelization. However, it has been observed in Table 4.4 that the maximum list length does not decrease inverse proportionally with the number of processors, and the numbers of objective function and constraint evaluations are relatively few, such performance is rather predictable. Figure 4.21, which displays the speedup of Problems 2 and 9, presents super-linear speedup cases.

To sum up, the blocking message passing model can ensure that the total effort (objective function and constraint evaluations) spent to solve a problem with parallel implementations almost identical with the serial computa-

4.5 Parallel Implementation

Figure 4.18: Speedup for Problems 1, 7 and 8 using blocking message passing.

Figure 4.19: Speedup for Problems 3, 6 and 10 using blocking message passing.

4.5 Parallel Implementation

Figure 4.20: Speedup for Problems 4 and 5 using blocking message passing.

Figure 4.21: Speedup for Problems 2 and 9 using blocking message passing.

tion model, which is important when the evaluations are costly and time-consuming. At the same time, the maximum list length, which indicates the work load distributed among all processors, decreases as the number of parallel processors increases. This effect is particularly prominent when the number of objective function and constraint evaluations is big. In such cases, management and storage of boxes may become a bottleneck if the work load is unevenly distributed. It can also be concluded that as the number of evaluations goes up, the chance of achieving more uniformly distributed work load is higher.

4.5.4 Non-Blocking Message Passing Model

In the blocking message passing model, the message transferring and box processing are executed in a sequential way based on the assumption that the processing phase is time-consuming. This will pose challenges when the time spend on box processing is short or when there is delay in message transferring. However, there exist non-blocking message passing operations in MPI, which enable communication overlap computation. This feature is important when the time spend on computation in each computation cycle is rather short, in which case the speedup will be greatly affected by the message passing operations.

In the non-blocking message passing model, the topology is the same as the blocking message passing model. Each processor sends messages to its right neighbor and receives messages from its left neighbor as the case in the blocking message passing model. A computation cycle consists of non-blocking message passing and computation operations. However, there is no gathering phase any more. Best known lower bound and processor

4.5 Parallel Implementation

status are sent together with the box information rather than calling an ALLGATHER operation. When the non-blocking message passing operations in previous computation cycles are not finished, the current computation cycle will only consist of box processing, thus, each computation cycle may consist of message passing operations or may not, depends on the status of previous message passing operations.

The messages sent between processors include box information when the sending processor has boxes for transferring, current best known lower bound and an array of processor status used for termination detection. A processor, denoted as P_0 , initiates the message passing flow and detects the termination condition. After a message is received, the status array is copied for next sending operation. The array of processor status keeps the status of each processor and is passed from left to right along the ring. Each processor can only set the status of itself and its right neighbor. When a box is packed to be sent, the status of current processor and its right neighbor are set to active; otherwise, the status of current processor is set to idle. It should be noted that the status change of a particular processor is not reflected among all processors at the same time, instead, it is passed one after another through message transferring. This prevents wrong termination detection.

Algorithm 4.5 introduces the non-blocking message passing model. The message sending flag F_s and message receiving flag F_r are used to check the status of non-blocking message passing operations. The status can either be SENT and WAITING for F_s and RECEIVED and WAITING for F_r . All processors start with a non-blocking message receiving operation except the processor P_0 , which first sends a message to its right neighbor, then starts the non-blocking message receiving operation. This initiates the message flow among

Algorithm 4.5 P-COPIA using non-blocking message passing

Require: $\mathcal{L}$, $\mathcal{B}_r$, $\mathcal{B}_s$, F_r , F_s ;**Ensure:** S ;

```

1: if  $P_0$  then
2:   pack the whole search domain as a box  $\mathcal{B}_s$ ;
3:   NB_SEND_MESSAGE ( $\mathcal{B}_s$ , right, BOX,  $F_s$ );
4: end if
5:
6: NB_RECV_MESSAGE ( $\mathcal{B}_r$ , left, RECVTAG,  $F_r$ );
7:
8: repeat
9:   check  $F_r$ ;
10:  if  $F_r$  is RECEIVED then
11:    Recv_Message_Processing ();
12:    NB_RECV_MESSAGE ( $\mathcal{B}_r$ , left, RECVTAG,  $F_r$ );
13:  end if
14:
15:  if  $\mathcal{L} = \emptyset$  and  $F_r$  is WAITING then
16:    wait until  $F_r$  is RECEIVED;
17:    Recv_Message_Processing ();
18:    NB_RECV_MESSAGE ( $\mathcal{B}_r$ , left, RECVTAG,  $F_r$ );
19:  end if
20:
21:  Box_Processing ();
22:
23:  if  $F_r$  is RECEIVED then
24:    Send_Message_Processing ();
25:    NB_SEND_MESSAGE ( $\mathcal{B}_s$ , right, SENDTAG,  $F_s$ );
26:  end if
27:
28:  check  $F_s$ ;
29:  if  $\mathcal{L} = \emptyset$  and  $F_s$  is WAITING then
30:    wait until  $F_s$  is SENT;
31:  end if
32: until  $\mathcal{B}_s$  status array on  $P_0$  are all idle
33:
34: if  $P_0$  then
35:   SEND_MESSAGE ( $\emptyset$ , right, QUIT);
36:   RECV_MESSAGE ( $\mathcal{B}_r$ , left, RECVTAG);
37: end if

```

4.5 Parallel Implementation

all processors. In each computation cycle, first the message receiving flag F_r is checked. If a message has arrived, the message is processed according to Algorithm 4.6 and another non-blocking receiving operation is started. Otherwise, if the working list $\mathcal{L}$ is empty, then the processor waits until a message is received and processes the message according to Algorithm 4.6 and starts another non-blocking receiving operation. In the processing phase, a box is selected and processed according to Algorithm 4.7 if the working list $\mathcal{L}$ is not empty. After that, the processor packs a box according to Algorithm 4.8 and sends the message to its right neighbor if a message has been received in this computation cycle. The message sending flag F_s is then checked, in case the working list $\mathcal{L}$ is empty and F_s is WAITING, the processor waits until this message sending operation finishes; otherwise, it just ends this computation cycle and starts a new one. The termination detection is done by processor P_0 . When the status of all processors in P_0 's sending status array are idle, P_0 sends a QUIT signal to its right neighbor P_1 . Consequently, the QUIT signal is passed from left to right to terminate all processors. The use of non-blocking message passing makes the procedure more complex than the blocking message passing model since even when all processors are currently idle, it is not sufficient to terminate the computation. There might be messages in transition which will reactivate an idle processor. However, the non-blocking receiving and sending are still pairwise in each processor, which ensures the messages to be matched and forms a continuous message flow within all processors.

Algorithm 4.6 describes the operation for processing received messages. First the status array is copied from received message to sending message status array. Then the messages are handled by looking at the RECVTAG. If

Algorithm 4.6 Recv_Message_Processing ()

```

1: copy status array from  $\mathcal{B}_r$  to  $\mathcal{B}_s$ ;
2: if RECVTAG is QUIT and not  $P_0$  then
3:   SEND_MESSAGE ( $\emptyset$ , right, QUIT);
4:   quit;
5: end if
6: if a better solution is received then
7:   update best known lower bound;
8:   update working list  $\mathcal{L}$ ;
9: end if
10: if RECVTAG is BOX then
11:   insert  $\mathcal{B}_r$  into  $\mathcal{L}$ ;
12: end if
13: if RECVTAG is IDLE then
14:   set left processor as idle in the status array;
15: end if

```

Algorithm 4.7 Box_Processing ()

```

1: if  $\mathcal{L} \neq \emptyset$  then
2:    $\mathcal{B} = \text{get}(\mathcal{L})$ ;
3:   subdivide  $\mathcal{B}$  into subsets  $\mathcal{B}_i$ ;
4:   for each  $\mathcal{B}_i$  do
5:     evaluate  $f(\mathcal{B}_i)$  and  $\mathbf{c}(\mathcal{B}_i)$ ;
6:     if better solution is found then
7:       update current best solution;
8:     end if
9:     if  $\mathcal{B}_i$  cannot be discarded then
10:      insert  $\mathcal{B}_i$  into  $\mathcal{L}$  in descending upper bound order;
11:     end if
12:   end for
13: end if

```

Algorithm 4.8 Send_Message_Processing ()

```

1: pack the best known lower bound;
2: if  $\mathcal{L} \neq \emptyset$  then
3:    $\mathcal{B} = \text{get}(\mathcal{L})$ ;
4:   SENDTAG = BOX;
5:   set current and right processors as non-idle in the status array;
6: else
7:   SENDTAG = IDLE;
8:   set current processor as idle in the status array;
9: end if

```

the tag is a QUIT tag, then the processor sends a QUIT tag to its right neighbor and exits. If a better known lower bound is received, the best known lower bound on this processor and the working list $\mathcal{L}$ are updated. If the RECVTAG is a BOX tag, the box is inserted into the working list $\mathcal{L}$ in descending order according to its upper bound.

Algorithm 4.7 shows the processing phase of a computation cycle. If the working list $\mathcal{L}$ is not empty then the first box of the list $\mathcal{L}$ is selected and partitioned; the child boxes are subsequently evaluated and either inserted back into $\mathcal{L}$ if they are pending or feasible, or discarded when they are infeasible or inferior to the best known solution. Of course, during box splitting, if a better solution is found, the current best solution and working list $\mathcal{L}$ need to be updated.

Algorithm 4.8 describes the operation for message sending. The best known lower bound on current processor is first packed into the message. If the working list $\mathcal{L}$ is not empty, the tag is set to BOX and a box is retrieved from $\mathcal{L}$ for sending, at the same time, the status of current processor and its right neighbor are set to non-idle in the status array; otherwise, the send tag is set to IDLE and the status of current processor is set to idle in the status array.

4.5.5 Non-Blocking Message Passing Performance

The performance of non-blocking message passing model is tested on both batches of problems in Table 4.2. Due to the non-blocking message passing mechanism, the number of objective and constraint functions evaluations may not be the same between different runs since the delay of message transferring might change the order of boxes to be processed. The test environment is the

4.5 Parallel Implementation

same as described in Section 4.5.3. The infeasibility index approach variant of P-COPIA is used for testing. The parameters ε_w , ε_f and ε for the first batch problems are described in Section 4.4.2. For the second batch problems, $\varepsilon_w = \varepsilon_f = \varepsilon = 10^{-2}$ for Problems 11, 14, 15 and 16, $\varepsilon_w = \varepsilon_f = \varepsilon = 10^{-3}$ for Problems 12 and 13.

Table 4.6 presents the numbers of objective and constraint functions evaluations using 1, 2, 4, 8 and 16 processors, respectively. It can be observed that in most cases, more evaluations are needed for the non-blocking message passing model. This is due to the delay in message transferring which results in unnecessary evaluations. However, some problems utilize less objective function and constraint evaluations when using the parallel model, which indicates that the parallel code takes a different search order from the serial model. Such effects are easier to be observed in the second batch of problems, which includes higher dimension problems. This might indicate that when the number of variables goes up, the parallel model can better utilize the alternative orders to process the boxes, which in turn may give an early detection of good solutions. However, it should be pointed out that the message transferring speed also affects the performance. This may account for the non-uniform increase or decrease in objective function and constraint evaluations using different number of processors, which are quite consistent in the blocking message passing model.

Table 4.7 presents the time used in seconds with 1, 2, 4, 8 and 16 processors, respectively. Figures 4.22 to 4.29 plot the speedups derived from Table 4.7. For the first batch problems except Problem 9, the time used by the parallel implementation is less than the serial model for at least once. However, the speedups deteriorate when shifting from 8 processors to 16

4.5 Parallel Implementation

Table 4.6: Objective and constraint evaluations using different number of processors.

Processor	1		2		4		8		16	
	fun.eva.	con.eva.	fun.eva.	con.eva.	fun.eva.	con.eva.	fun.eva.	con.eva.	fun.eva.	con.eva.
1	169554	815690	172130	827880	179393	862417	216143	1043376	270125	1301613
2	59372	437082	64854	472753	52847	397302	64240	477683	137727	990849
3	23637	81993	29897	101887	45946	156407	63686	199058	92098	277415
4	146	566	154	580	177	644	155	583	155	583
5	263	365	273	371	273	371	273	371	267	371
6	13752	50830	19750	73164	31872	117719	40909	151773	52858	189193
7	3120	51	4685	369	3874	648	3670	408	2520	366
8	80335	295577	106862	385873	146427	508134	183731	642117	243481	821020
9	54138	96483	59053	104089	29630	53623	22941	42755	23124	43130
10	9875	196	10497	205	10995	253	11696	265	14462	319
11	693668	31805	497564	118169	192572	138600	130421	249322	219501	304920
12	365081	2765102	361237	2738602	322973	2484645	326116	2504490	357013	2720016
13	639484	2204374	571496	2068831	684017	2477801	1086901	4069367	1104110	4348840
14	309684	507397	358220	586906	335139	551975	282347	459382	286554	467326
15	1866677	3044586	214400	3500170	1798656	2977285	1766403	2904029	1610199	2649758
16	1181529	23662	1850162	22048079	2120874	18325953	562662	4096324	879626	733156

4.5 Parallel Implementation

Table 4.7: Times used with different number of processors (in seconds).

<i>Problem</i>	<i>1</i>	<i>2</i>	<i>4</i>	<i>8</i>	<i>16</i>
1	50.42	13.76	6.90	3.78	10.73
2	5.75	1.24	0.68	0.84	1.78
3	0.58	0.31	0.31	0.38	0.52
4	0.003	0.003	0.005	0.009	0.01
5	0.007	0.006	0.008	0.03	0.02
6	0.37	0.25	0.23	0.26	0.25
7	0.05	0.03	0.04	0.03	0.05
8	2.89	1.37	1.22	1.14	1.89
9	4.27	1.46	0.38	0.22	0.18
10	0.05	0.04	0.05	0.07	0.05
11	4.66	1.40	0.46	0.25	0.20
12	496.25	70.90	21.51	13.74	8.07
13	596.03	121.97	37.10	22.95	21.52
14	306.92	91.29	18.64	5.12	4.01
15	11157.48	3030.48	480.36	121.98	47.49
16	696.12	248.85	102.81	6.53	0.82

processors; sometimes even occur at an earlier stage, that is, from 4 processors thereafter. This suggests that the extra cost spent on communication is greater than the gain by taking more processors in these problems. Moreover, when such cases happen, it can be observed from Table 4.6 that the numbers of evaluations are bigger than the serial model in most cases.

It can be observed from Figure 4.22 that the speedups of Problems 1 and 8 decline after 8 processors while Problem 2 degenerates from 4 processors. The speedups of Problems 3 and 6 shown in Figure 4.23 show no improvement after utilizing 2 processors. The performance of Problems 4, 5, 7 and 10 plotted in Figures 4.24 and 4.25 are rather bad since the speedup fluctuates around 1.0 and even less than 1.0 in some cases irrespective of number of processors used. However, considering the time needed to solve these problems (less than 0.06 second even with a serial implementation), the performance

4.5 Parallel Implementation

Figure 4.22: Speedup for Problems 1, 2 and 8 using non-blocking message passing.

Figure 4.23: Speedup for Problems 3 and 6 using non-blocking message passing.

4.5 Parallel Implementation

Figure 4.24: Speedup for Problems 4 and 5 using non-blocking message passing.

Figure 4.25: Speedup for Problems 7 and 10 using non-blocking message passing.

4.5 Parallel Implementation

Figure 4.26: Speedup for Problems 9, 11 and 13 using non-blocking message passing.

Figure 4.27: Speedup for Problems 12 and 14 using non-blocking message passing.

4.5 Parallel Implementation

Figure 4.28: Speedup for Problem 15 using non-blocking message passing.

Figure 4.29: Speedup for Problem 16 using non-blocking message passing.

is rather reasonable.

The performance of Problem 9 and the second batch problems give much better results, all 7 problems exhibit super-linear speedups. Among them, Problems 12, 14, 15 and 16 exhibit quite impressive efficiency. A common characteristic of those problems is that they are more difficult and time-consuming to solve, which is exactly where parallel processing can help. The numerical results imply that the parallel model can not only speedup the computation, but also explores the search space in a much more efficient way.

4.6 Summary

In this chapter, three interval subdivision direction selection strategies are proposed and applied for constrained optimization. The strategies improve the performance by exploiting the currently available information, which is a must for solving the problem. Consequently, the parallel models for constrained optimization using interval analysis are presented. Two message passing mechanisms are investigated: blocking message passing and non-blocking message passing. The model with blocking message passing can ensure that the total effort used by the parallel implementation is almost equal to the serial model, but it is affected by the time spent on message passing. When the time spent on box processing is time-consuming, it can achieve an almost linear speedup. When the time used for communication occupies a high percentage in a computation cycle, the blocking message passing model may lose its advantage. The non-blocking message passing model, which makes communication overlap computation, is quite efficient when the box processing phase is not time-consuming. However, it might result in more

4.6 Summary

objective function and constraint evaluations. But it is also possible that the model explores the search space in a more efficient way which results in super-linear speedup. Numerical experiments indicate that P-COPIA with non-blocking message passing performs better when the problems are more difficult to solve.

Chapter 5

Conclusions and Future Recommendations

5.1 Conclusions

Optimization techniques play an important role both in theory and real engineering applications. In general, the global optimum solution is wanted when the objective function and constraints are non-convex. However, this is a difficult task to accomplish, especially for highly multimodal problems. Moreover, in many problems, the function is expensive and time-consuming to evaluate, and additional information such as derivative is difficult to obtain. Under these assumptions, derivative-free methods which seek to minimize the effort to solve problems are investigated. At the same time, the fast developing computation technologies and equipment provide an opportunity for parallel processing, which shortens the time needed to solve problems when parallel computation is properly applied.

This research investigates both methods to solve global optimization problems and models to parallelize those methods. Besides, all approaches only utilize the objective function and constraints information, and no additional knowledge such as the derivatives is included. The advantage of those

methods is that they are still applicable to problems where the information of derivative is costly to evaluate or simply not available.

For unconstrained optimization, the method, TRIOPT, which employs simplex as the basic partitioning element, is devised. Simplex is suitable for partitioning because of its high splitting performance. Combining the specially designed selection rules and adaptive partitioning schemes for simplex, TRIOPT provides a novel global optimization algorithm for box constrained problems.

TRIOPT is especially useful when some scattered samples are given in the very beginning of the search, that is, the algorithm is able to generate partitions from the given set of samples by using triangulation as the initial partitioning technique. It might be difficult for some algorithms to utilize this kind of information. One more advantage of TRIOPT is that it makes no assumption of the objective function, thus it can carry out its search under very mild conditions for black box optimization.

TRIOPT makes use of both local and global information to guide its search. The local information locates the new samples, while the global information indicates the place where to search. By updating the transformed value and removal of small simplices in each iteration, the algorithm conducts its search in a parallel manner with adaptive concurrency.

For constrained optimization, interval analysis is introduced since it can easily check the feasibility of a certain search area. Three strategies which do not utilize the derivative information are proposed and proved to be helpful for subdivision direction selection for constrained problems.

The first strategy—information gain, aims to reduce the upper bound of the objective function as quickly as possible. This sounds strange for

maximization problems. Further thinking indicates this operation makes the gap between the best upper bound and the best lower bound smaller, which leads to a convergent solution. The other two strategies—constraint violation approach and infeasibility index approach, are based on the information of constraints. The aim now becomes to make the constraint feasible as quickly as possible. This is reasonable because solutions of constrained problems must be feasible first. Since only the objective function information is used for the first strategy, it can also be applied to unconstrained optimization.

The three strategies effectively utilize and make their decisions based on currently available information. The information gain approach uses the “heritage” inherit from previous splits. The constraint violation and infeasibility index approaches simply look at the current constraints situation of the box. Thus, the decision made is adaptive to current box information. The global selection of boxes and the local adaptiveness of subdivision direction work together provide efficient approaches for verified constrained optimization.

Parallel computation models are proposed for both TRIOPT and COPIA according to their algorithmic characteristics. Due to the dynamic degree of parallelization, TRIOPT employs a Master/Slave architecture. The adaptive concurrency enables the parallel model suitable for any number of processors. For COPIA, the branch and bound approach makes it easily convertible to parallel implementation. The advantage of the models presented in this research is that there is no need to design a special initial partition; no need to consider the load balancing problem; and the management of communication is very simple. Since there are no special assumptions about the problems under consideration, the parallel models can be applied to general branch

5.2 Recommendations for Future Research

and bound problems.

The contribution of this research can be briefly summarized as follows:

- A dynamic parallelism algorithm, TRIOPT, is proposed for black box unconstrained optimization problems. The high splitting performance and adaptive partition strategies enable the method to search effectively and efficiently in the domain.
- Three interval subdivision strategies are presented. All methods make use of information derived from the problem itself and no additional information other than the function ranges is needed.
- The parallel computation models proposed for constrained optimization can be used for general branch and bound problems. Any number of processors can be adapted for the parallel model. Moreover, the dispense with initial partition and load balancing, plus easy management of communication make the parallel implementation as easy as a serial one.

5.2 Recommendations for Future Research

Recommendations for future research that merits further investigation may include:

In unconstrained optimization, TRIOPT needs the vertices of the search domain to enable the triangulation to cover the whole search area; therefore, TRIOPT is limited to low dimensional problems. For high dimensional problems, partitions might need to be represented with incomplete information; that is, a partition is assessed with only a few vertices instead of all vertices.

5.2 Recommendations for Future Research

This idea can be expanded similarly to rectangular partitioning schemes. Nevertheless, the generic search strategy proposed in TRIOPT can still be applied to employ the dynamic degree of parallelism. Specific methods may need modifications while the basic idea should be kept unchanged.

For constrained optimization problems using interval analysis, it might be helpful to incorporate the constraints information into the box selection criteria due to the fact that in a branch and bound technique, the order of branches to be processed is also very important. In constrained optimization, normally the global optimum of objective function is not feasible due to the constraints. Therefore, taking the objective function as the sole selection criterion might not be the optimal strategy. Further, incorporating the constraints into the selection criteria enables a procedure to find a feasible solution quickly, which can subsequently be used to cut off the branches that are inferior to it and eventually speeds up the processing. Fuzzy assessment can be one possibility to be used to represent the feasibility of a box in determining the order of the branches to be processed. A potential approach might be first to transform the feasibility interval of a single constraint into a fuzzy number and then use an aggregated value for all constraints to decide which box should be processed first. This might eliminate scaling problems in the assessment measure.

The past decade has witnessed a flurry of interest within optimization regarding artificial intelligence technologies, including neural networks, fuzzy systems, and genetic algorithms (GA). In many ways, GA and the extension of genetic programming, may offer an outstanding combination of flexibility, robustness, and simplicity. It could be a fruitful future research area in estimation of global optima if basic GA concepts, such as mutation and

5.2 Recommendations for Future Research

crossover, are incorporated with the proposed algorithms in the thesis.

For the parallel model with blocking message passing, although the performance with up to 16 processors is quite good, it might not be a good choice when the number of processors becomes large. Methods which can distribute subproblems to all processors in a faster way are required. One possibility might be the alternative message passing, that is, instead of always sending messages to the immediate right neighbors, communications are established with other processors which are sometimes “far away”. Similarly, the sequential passing of optimal solution and boxes in the non-blocking message passing model may also not be efficient when the number of processors becomes large. Alternatives are also wanted to speedup the passing of best known solutions. However, in this case, the termination criterion should be re-designed since the alternation of message passing order affects the termination detection.

References

- Aurenhammer, F. (1991). Voronoi diagrams—a survey of a fundamental geometric data structure. *ACM Computing Surveys (CSUR)*, 23(3):345–405.
- Bäck, T. (1996). *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, New York.
- Barber, C. B., Dobkin, D. P., and Huhdanpaa, H. (1996). The Quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software*, 22(4):469–483.
- Benson, H. P. (1995). Concave minimization: theory, applications, and algorithms. In Horst, R. and Pardalos, P. M., editors, *Handbook of Global Optimization*, pages 43–148. Kluwer Academic Publishers, Dordrecht / Boston / London.
- Benyoub, A. and Daoudi, E. M. (2001). Parallelization of the continuous global optimization problem with inequality constraints by using interval arithmetic. In *Proceedings of the 9th International Conference on High-Performance Computing and Networking*, pages 595–602. Springer-Verlag.

REFERENCES

- Bern, M. W. and Eppstein, D. (1992). Mesh generation and optimal triangulation. In Du, D.-Z. and Hwang, F. K.-M., editors, *Computing in Euclidean Geometry*, pages 23–90. World Scientific, Singapore.
- Berner, S. (1996). New results on verified global optimization. *Computing*, 57:323–343.
- Björkman, M. and Holmström, K. (2000). Global optimization of costly nonconvex functions using radial basis functions. *Optimization and Engineering*, 1:373–397.
- Boender, G. G. E. and Romeijn, H. E. (1995). Stochastic methods. In Horst, R. and Pardalos, P. M., editors, *Handbook of Global Optimization*, pages 829–869. Kluwer Academic Publishers, Dordrecht / Boston / London.
- Carter, R. G., Gablonsky, J. M., Patrick, A., Kelley, C. T., and Eslinger, O. J. (2001). Algorithms for noisy problems in gas transmission pipeline optimization. *Optimization and Engineering*, 2:139–157.
- Casado, L. G., García, I., and Csendes, T. (2001a). A heuristic rejection criterion in interval global optimization algorithms. *BIT Numerical Mathematics*, 41(4):683–692.
- Casado, L. G., Martínez, J. A., and García, I. (2001b). Experiments with a new selection criterion in a fast interval optimization algorithm. *Journal of Global Optimization*, 19:247–264.
- Church, R. L. (2002). Geographical Information Systems and location science. *Computers & Operations Research*, 29:541–562.

REFERENCES

-
- Clausen, J. and Žilinskas, A. (2002). Subdivision, sampling, and initialization strategies for simplicial branch and bound in global optimization. *Computers and Mathematics with Applications*, 44:943–955.
- Coope, I. D. and Price, C. J. (2001). On the convergence of grid-based methods for unconstrained optimization. *SIAM Journal on Optimization*, 11(4):859–869.
- Csallner, A. E., Csendes, T., and Markót, M. C. (1999). Multisection in interval branch-and-bound methods for global optimization I: Theoretical results. *Journal of Global Optimization*, 16:371–392.
- Csendes, T. (2001). New subinterval selection criteria for interval global optimization. *Journal of Global Optimization*, 19:307–327.
- Csendes, T. and Ratz, D. (1997). Subdivision direction selection in interval methods for global optimization. *SIAM Journal on Numerical Analysis*, 34(3):922–938.
- Cuillière, J. C. (1998). An adaptive method for the automatic triangulation of 3D parametric surfaces. *Computer-Aided Design*, 30(2):139–149.
- Demirhan, M. B. and Özdamar, L. (1999). A note on the use of a fuzzy approach in adaptive partitioning algorithms for global optimization. *IEEE Transactions on Fuzzy Systems*, 7:468–475.
- Demirhan, M. B. and Özdamar, L. (2003). A fuzzy adaptive partition algorithm (FAPA) for global optimization. In Verdegay, J.-L., editor, *Fuzzy Sets Based Heuristics for Optimization*, volume 126 of *Studies in Fuzziness and Soft Computing*, pages 37–48. Springer-Verlag, Berlin.

REFERENCES

- Demirhan, M. B., Özdamar, L., Helvacioğlu, L., and Birbil, S. I. (1999). FRACTOP: A geometric partitioning metaheuristic for global optimization. *Journal of Global Optimization*, 14:415–436.
- Dennis Jr., J. E. and Torczon, V. (1991). Direct search methods on parallel machines. *SIAM Journal on Optimization*, 1(4):448–474.
- Deutsch, C. V. and Journel, A. G. (1998). *GSLIB: Geostatistical Software Library and User's Guide*. Oxford University Press, New York.
- Dolan, E. D., Lewis, R. M., and Torczon, V. (2000). On the local convergence of pattern search. Technical report, Institute for Computer Applications in Science and Engineering, NASA Langley Research Center, Hampton, VA.
- Floudas, C. A., Pardalos, P. M., Adjiman, C. S., Esposito, W. R., Günius, Z. H., Harding, S. T., Klepeis, J. L., Meyer, C. A., and Schweiger, C. A. (1999). *Handbook of Test Problems in Local and Global Optimization*. Kluwer Academic Publishers, Dordrecht / Boston / London.
- Flynn, M. J. (1972). Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, C-21(9):948–960.
- Gau, C.-Y. and Stadtherr, M. A. (2001). Parallel interval-Newton using message passing: dynamic load balancing strategies. In *Supercomputing '01: Proceedings of the 2001 ACM/IEEE Conference on Supercomputing (CDROM)*, pages 23–23, New York, NY, USA. ACM Press.
- Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley.

REFERENCES

- Gourdin, E., Hansen, P., and Jaumard, B. (1994). *Global optimization of multi-variate Lipschitz functions: a survey and computational comparison*. Les Cahiers du GERAD. McGill University, Montreal.
- Gutmann, H.-M. (2001). A radial basis function method for global optimization. *Journal of Global Optimization*, 19(3):201–227.
- Hammer, R., Hocks, M., Kulisch, U., and Ratz, D. (1995). *C++ Toolbox for Verified Computing*. Springer-Verlag, Berlin.
- Hansen, E. (1992). *Global Optimization Using Interval Analysis*. Marcel Dekker, New York.
- Hansen, E. and Sengupta, S. (1980). Global constrained optimization using interval analysis. In Nickel, K., editor, *Interval Mathematics*, pages 25–47. Springer-Verlag, Berlin.
- Hansen, P. and Jaumard, B. (1995). Lipschitz optimization. In Horst, R. and Pardalos, P. M., editors, *Handbook of Global Optimization*, pages 407–493. Kluwer Academic Publishers, Dordrecht.
- He, J., Watson, L. T., Ramakrishnan, N., Shaffer, C. A., Verstak, A., Jiang, J., Bae, K., and Tranter, W. H. (2002). Dynamic data structures for a direct search algorithm. *Computational Optimization and Applications*, 23:5–25.
- Henriksen, T. and Madsen, K. (1992). Use of a depth-first strategy in parallel global optimization. Technical Report 92-10, Institute of Numerical Analysis, Technical University of Denmark, Lyngby.

REFERENCES

- Hooke, R. and Jeeves, T. A. (1961). “Direct Search” solutions of numerical and statistical problems. *Journal of the Association of Computing Machinery*, 8:212–229.
- Horst, R. and Pardalos, P. M. (1995). *Handbook of Global Optimization*. Kluwer, Dordrecht.
- Horst, R., Pardalos, P. M., and Thoai, N. V. (1995). *Introduction to Global Optimization*. Kluwer, Dordrecht.
- Horst, R. and Tuy, H. (1996). *Global Optimization. Deterministic Approaches*. Springer Verlag, Berlin.
- Hough, P. D., Kolda, T. G., and Torczon, V. J. (2002). Asynchronous parallel pattern search for nonlinear optimization. *SIAM Journal on Scientific Computing*, 23(1):134–156.
- Huyer, W. and Neumaier, A. (1999). Global optimization by multilevel coordinate search. *Journal of Global Optimization*, 14:331–355.
- Ibraev, S. (2001). *A new parallel method for verified global optimization*. PhD Thesis, University of Wuppertal, Germany.
- Ingber, L. (1993). Simulated Annealing: Practice versus theory. *Mathematical and Computer Modelling*, 18(11):29–57.
- Jones, D. R. (1996). Global optimization with response surfaces. In *Fifth SIAM Conference on Optimization*, Canada.
- Jones, D. R. (2001). A taxonomy of global optimization methods based on response surfaces. *Journal of Global Optimization*, 21:345–383.

REFERENCES

- Jones, D. R., Perttunen, C. D., and Stuckman, B. E. (1993). Lipschitzian optimization without the Lipschitz constant. *Journal of Optimization Theory and Applications*, 79:157–181.
- Jones, D. R., Schonlau, M., and Welch, W. J. (1998). Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13:455–492.
- Kearfott, R. B. (1996a). *Rigorous Global Search: Continuous Problems*. Kluwer, Dordrecht.
- Kearfott, R. B. (1996b). Test results for an interval branch and bound algorithm for equality-constrained optimization. In Floudas, C. A. and Pardalos, P. M., editors, *State of the Art in Global Optimization: Computational Methods and Applications*, pages 181–200. Kluwer Academic Publisher, Berlin.
- Kirkpatrick, S., Gelatt, C. D., and Jr. Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220(4598):671–680.
- Knüppel, O. (1994). PROFILE/BIAS — a fast interval library. *Computing*, 53:277–287.
- Kolda, T. G., Lewis, R. M., and Torczon, V. (2003). Optimization by direct search: new perspectives on some classical and modern methods. *SIAM Review*, 45(3):385–482.
- Krikanov, A. A. (2000). Composite pressure vessels with higher stiffness. *Composite Structures*, 48:119–127.

REFERENCES

- Lagarias, J. C., Reeds, J. A., Wright, M. H., and Wright, P. E. (1998). Convergence properties of the nelder-mead simplex method in low dimensions. *SIAM Journal on Optimization*, 9(1):112–147.
- Lattuada, R. and Raper, J. (1996). Application of 3D Delaunay triangulation algorithms in geoscientific modelling. In *Proceedings of the Third International Conference/Workshop on Integrating GIS and Environmental Modeling*, Santa Fe, New Mexico, USA.
- Lewis, R. M. and Torczon, V. (1999). Pattern search methods for bound constrained minimization. *SIAM Journal on Optimization*, 9(4):1082–1099.
- Lewis, R. M., Torczon, V., and Trosset, M. W. (2000). Direct search methods: Then and now. *Journal of Computational and Applied Mathematics*, 124:191–207.
- Mäkelä, M. M. and Neittaanmäki, P. (1992). *Nonsmooth Optimization*. World Scientific, Singapore / New Jersey / London / Hong Kong.
- Markót, M. C., Fernández, J., Casado, L. G., and Csendes, T. New interval methods for constrained global optimization. *Mathematical Programming*. To appear.
- McKinnon, K. I. M. (1998). Convergence of the Nelder-Mead simplex method to a nonstationary point. *SIAM Journal on Optimization*, 9(1):148–158.
- Message Passing Interface Forum (1994). MPI: The message passing interface standard.
- Meza, J. C. (1994). OPT++: An object-oriented class library for nonlinear optimization. Technical Report Report No. SAND94-8225, Scientific Computing Department, Sandia National Laboratories, California.

REFERENCES

- Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, Berlin / Heidelberg / New York, 3rd edition.
- Moore, R. E. (1966). *Interval Analysis*. Prentice Hall, Englewood Cliffs.
- Moore, R. E. and Ratschek, H. (1988). Inclusion functions and global optimization II. *Mathematical Programming*, 41:341–356.
- Nataraj, P. S. V. and Sheela, S. (2002). A new subdivision strategy for range computations. *Reliable Computing*, 8:83–92.
- Nelder, J. A. and Mead, R. (1965). A simplex method for function minimization. *Computer Journal*, 7(4):308–313.
- Neumaier, A. (2004). Complete search in continuous global optimization and constraint satisfaction. In Iserles, A., editor, *Acta Numerica 2004*. Cambridge University Press.
- Özdamar, L. and Demirhan, M. B. (2000). Experiments with new stochastic global optimization search techniques. *Computers & Operations Research*, 27:841–865.
- Özdamar, L. and Demirhan, M. B. (2001). Comparison of partition evaluation measures in an adaptive partitioning algorithm for global optimization. *Fuzzy Sets and Systems*, 117(1):47–60.
- Özdamar, L. and Demirhan, M. B. (2003). A fuzzy adaptive partitioning algorithm (FAPA) for global optimization: Implementation in environmental site characterization. In Verdegay, J.-L., editor, *Fuzzy Sets Based Heuristics for Optimization*, volume 126 of *Studies in Fuzziness and Soft Computing*, pages 307–316. Springer-Verlag, Berlin.

REFERENCES

- Pal, N. R. and Pal, S. K. (1989). Object background segmentation using new definitions of entropy. *IEE Proceedings-E*, 136(4):284–295.
- Pintér, J. (1988). Branch and bound algorithms for solving global optimization problems with lipschitzian structure. *Optimization*, 19:101–110.
- Pintér, J. (1992). Convergence qualification of adaptive partition algorithms in global optimization. *Mathematical Programming*, 56:343–360.
- Pintér, J. (1996). *Global Optimization in Action*. Kluwer Academic Publishers, Dordrecht.
- Pintér, J. (1997). LGO—A program system for continuous and Lipschitz global optimization. In Bomze, I. M., Csendes, T., Horst, R., and Pardalos, P. M., editors, *Developments in Global Optimization*, pages 183–197. Kluwer Academic Publishers, Dordrecht / Boston / London.
- Pintér, J. (2002). Model development, solution, and analysis in global optimization. In Pintér, J., editor, *Global Optimization—Selected Case Studies*. Kluwer Academic Publishers, Dordrecht / Boston / London.
- Pintér, J. (2003). *LGO: A Model Development and Solver System for Continuous Global Optimization User Guide*. Pinter Consulting Services, Halifax, Nova Scotia, Canada.
- Powell, M. J. D. (1964). An efficient method for finding the minimum of a function of several variables without calculating derivatives. *Computer Journal*, 7(2):155–162.
- Powell, M. J. D. (1992). The theory of radial basis function approximation in 1990. In Light, W., editor, *Wavelets, Subdivision Algorithms and Radial*

REFERENCES

- Basis Functions*, volume 2 of *Advances in Numerical Analysis*, pages 105–210. Oxford University Press.
- Pownuk, A. (2000). Optimization of mechanical structures using interval analysis. *Computer Assisted Mechanics and Engineering Science*, 7(4):699–705.
- Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (1992). *Numerical Recipes in C*. Cambridge University Press, 2nd edition.
- Price, C. J., Coope, I. D., and Byatt, D. (2002). A convergent variant of the Nelder-Mead algorithm. *Journal of Optimization Theory and Applications*, 113:5–19.
- Price, W. L. (1977). A controlled random search procedure for global optimization. *Computer Journal*, 20(4):367–370.
- Ratschek, H. and Rokne, J. (1988). *New Computer Methods for Global Optimization*. Ellis Horwood Limited, New York / Chichester / Brisbane / Toronto.
- Ratschek, H. and Rokne, J. (1995). Interval methods. In Horst, R. and Pardalos, P. M., editors, *Handbook of Global Optimization*, pages 751–828. Kluwer Academic Publishers, Dordrecht / Boston / London.
- Ratz, D. (1992). *Automatische Ergebnisverifikation bei globalen Optimierungsproblemen*. PhD Thesis, Universität Karlsruhe, Karlsruhe, Germany.
- Ratz, D. and Csendes, T. (1995). On the selection of subdivision directions in interval branch-and-bound methods for global optimization. *Journal of Global Optimization*, 7:183–207.

REFERENCES

- Rosenbrock, H. H. (1960). An automatic method for finding the greatest or least value of a function. *Computer Journal*, 3(3):175–184.
- Schumaker, L. L. (1993). Triangulations in CAGD. *Computer Graphics and Applications, IEEE*, 13(1):47–52.
- Schwefel, H.-P. (1981). *Numerical Optimization of Computer Models*. Wiley & Sons, Chichester / New York.
- Shubert, B. (1972). A sequential method seeking a global maximum of a function. *SIAM Journal on Numerical Analysis*, 9:379–388.
- Skelboe, S. (1974). Computation of rational interval functions. *BIT*, 14:87–95.
- Stillinger, F. H., Head-Gordon, T., and Hirshfeld, C. L. (1993). Toy model for protein folding. *Physical Review E*, 48(2):1469–1477.
- Su, P. and Drysdale III, R. L. S. (1995). A comparison of sequential delaunay triangulation algorithms. In *Symposium on Computational Geometry*, pages 61–70.
- Tang, Z. B. (1994). Adaptive partitioned random search to global optimization. *IEEE Transactions on Automatic Control*, 39(11):2235–2244.
- Torczon, V. (1997). On the convergence of pattern search algorithms. *SIAM Journal on Optimization*, 7(1):1–25.
- Törn, A. and Žilinskas, A. (1989). *Global Optimization, Vol. 350 of Lecture Notes in Computer Science*. Springer-Verlag, Berlin.
- Trosset, M. W. (1997). I know it when I see it: Toward a definition of direct search methods. *SIAG/OPT Views-and-News*, 9:7–10.

REFERENCES

- Tseng, P. (1999). Fortified-descent simplicial search method: A general approach. *SIAM Journal on Optimization*, 10:269–288.
- Vavasis, S. A. (1995). Complexity issues in global optimization: A survey. In Horst, R. and Pardalos, P. M., editors, *Handbook of Global Optimization*, pages 27–41. Kluwer Academic Publishers, Dordrecht / Boston / London.
- Walters, F. H., Parker, Jr., L. R., Morgan, S. L., and Deming, S. N. (1991). *Sequential Simplex Optimization*. Chemometrics Series. CRC Press, Inc., Boca Raton, Florida.
- Watson, L. T. and Baker, C. A. (2001). A fully-distributed parallel global search algorithm. *Engineering Computations*, 18:155–169.
- Wolfe, M. A. (1994). An interval algorithm for constrained global optimization. *Journal of Computational and Applied Mathematics*, 50:605–612.
- Wood, G. (1991). Multidimensional bisection applied to global optimization. *Computers and Mathematics with Applications*, 21:161–172.
- Wright, M. H. (1996). Direct search methods: Once scorned, now respectable. In Griffiths, D. F. and Watson, G. A., editors, *Numerical Analysis 1995*, pages 191–208. Addison-Wesley Longman, Harlow, United Kingdom.
- Yu, X., Morse, B. S., and Sederberg, T. W. (2001). Image reconstruction using data-dependent triangulation. *Computer Graphics and Applications, IEEE*, 21(3):62–68.
- Zabinsky, Z. B. (1998). Stochastic methods for practical global optimization. *Journal of Global Optimization*, 13:433–444.

REFERENCES

- Zabinsky, Z. B. and Smith, R. L. (1992). Pure adaptive search in global optimization. *Mathematical Programming*, 53:323–338.
- Zabinsky, Z. B., Smith, R. L., McDonald, F., Romeijn, H. E., and Kaufman, D. E. (1993). Improving Hit-and-Run for global optimization. *Journal of Global Optimization*, 3:171–192.
- Zhang, B., Wood, G., and Baritompa, W. (1993). Multi-dimensional bisection: the performance and the context. *Journal of Global Optimization*, 3:337–358.

Appendix A

Test Problems for TRIOPT

In this appendix, the description of test problems for unconstrained optimization is given. Some surface plots of theoretical problems are also depicted here.

A.1 Theoretical Problems

Six-Hump

$\mathbb{R}^2$, GEATbx¹. Global and local optima are on a flat surface, the difference between global optima and local ones is small.

$$\begin{aligned} \max f(\mathbf{x}) &= -4x_1^2 + 2.1x_1^4 - x_1^6/3 - x_1x_2 + 4x_2^2 - 4x_2^4 \\ (x_1 &= [-2.0, 2.0], x_2 = [-2.0, 2.0]) \end{aligned}$$

G & P

$\mathbb{R}^2$, GEATbx¹. Very big function range, flat surface, deceptive function.

¹Genetic and Evolutionary Algorithm Toolbox for Matlab.
http://www.systemtechnik.tu-ilmenau.de/~pohlheim/GA_Toolbox/fcnindex.html

A.1 Theoretical Problems

Figure A.1: Surface plot of Six-Hump.

Figure A.2: Surface plot of G & P.

$$\begin{aligned} \max f(\mathbf{x}) = & -1(1 + (x_1 + x_2 + 1)^2 \\ & (19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)) \\ & (30 + (2x_1 - 3x_2)^2 \\ & (18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)) \\ & (x_1 = [-2.0, 2.0], x_2 = [-2.0, 2.0]) \end{aligned}$$

Figure A.3: Surface plot of Ackley.

Ackley

$\mathbb{R}^2$, Bäck (1996). Symmetric multi-modal test problem with evenly distributed maxima. The global optimum is at the center of original feasible region, therefore, the original feasible region is shifted to avoid a trivial solution.

$$\begin{aligned} \max f(\mathbf{x}) &= 20 \exp \left(-0.2 \sqrt{0.5(x_1^2 + x_2^2)} \right) \\ &\quad + \exp(0.5(\cos(2\pi x_1) + \cos(2\pi x_2))) - 20.0 - e \\ (x_1 &= [-1.5, 2.5], x_2 = [-2.0, 2.0]) \end{aligned}$$

Spike

$\mathbb{R}^2$, Michalewicz (1996). Two dimensional problem with innumerable local optima, haystack type, very narrow basin of attraction for global optimum, low difference between global and second best optimum.

$$\begin{aligned} \max f(\mathbf{x}) &= 21.5 + x_1 \sin(4\pi x_1) + x_2 \sin(20\pi x_2) \\ (x_1 &= [-3.0, 12.1], x_2 = [4.1, 5.8]) \end{aligned}$$

Figure A.4: Surface plot of Spike.

Figure A.5: Surface plot of Schwefel.

Schwefel

$\mathbb{R}^2$, Schwefel (1981). Deceptive function, global optimum near domain corner, three second best solutions very far away from the global optimum.

$$\max f(\mathbf{x}) = \sum_{i=1}^2 x_i \sin(\sqrt{|x_i|}) - 837.9658$$

$$(x_i = [-500.0, 500.0], i = 1, 2)$$

Figure A.6: Surface plot of Easom.

Figure A.7: Surface plot of Crescent.

Easom

$\mathbb{R}^2$, GEATbx². A unimodal test problem with very flat surface outside the basin of attraction.

$$\max f(\mathbf{x}) = \cos(x_1) \cos(x_2) \exp(-(x_1 - \pi)^2 - (x_2 - \pi)^2)$$

$$(x_i = [0.0, 6.0], i = 1, 2)$$

²Genetic and Evolutionary Algorithm Toolbox for Matlab.
http://www.systemtechnik.tu-ilmenau.de/~pohlheim/GA_Toolbox/fcnindex.html

Figure A.8: Surface plot of Griewank(2).

Crescent

$\mathbb{R}^2$, Mäkelä and Neittaanmäki (1992). Non-differentiable, a circular loci on which a singular global optimum lies on a crater-like basin of attraction, smooth elsewhere.

$$\begin{aligned} \max f(\mathbf{x}) = & -\max\{x_1^2 + (x_2 - 1)^2 + x_2 - 1, \\ & -x_1^2 - (x_2 - 1)^2 + x_2 + 1\} \\ & (x_1 = [-1.5, 1.5], x_2 = [-0.5, 2.5]) \end{aligned}$$

Griewank

$\mathbb{R}^n$, $n = 2, \dots, 4$, Törn and Žilinskas (1989). Many widespread evenly distributed local optima, increased complexity due to reduced original domain. Original feasible domain shifted.

$$\begin{aligned} \max f(\mathbf{x}) = & -\frac{1}{4000} \sum_{i=1}^n x_i^2 + \prod_{i=1}^n \cos\left(\frac{x_i}{\sqrt{i}}\right) - 1.0 \\ & (x_i = [-9.0, 11.0], i = 1, \dots, n) \end{aligned}$$

Figure A.9: Surface plot of Rastrigin(2).

Rastrigin

$\mathbb{R}^n$, $n = 2, \dots, 5$, Törn and Žilinskas (1989). Highly multi-modal, Evenly spaced haystack type, spherical and symmetric. Original feasible domain shifted.

$$\max f(\mathbf{x}) = \sum_{i=1}^n [10 \cos(2\pi x_i) - x_i^2] - 10n$$

$$(x_i = [-3.12, 5.12], i = 1, \dots, n)$$

A.2 Engineering Applications

Several applications from different fields compose the engineering applications test bed. They are briefly described below.

ANVCS

The ANVCS problem is a QFT bounds calculation problem. The Quantitative Feedback Theory (QFT) is an engineering design technique for uncertain feedback systems where closed-loop frequency-domain specifications are

translated into Nichols chart domains specifying the allowable range of the nominal open-loop response. In the maximization problem considered below, an upper bound for an active noise and vibration control system (ANVCS) (Nataraj and Sheela, 2002) is sought.

$$\max f(\mathbf{x}) = -\frac{180}{\pi} \arctan \left[\left(2x_2 \frac{\omega}{x_1} \right) / \left(1 - \left(\frac{\omega}{x_1} \right)^2 \right) \right]$$

$$x_1 = [0.75, 1.25], x_2 = [0.02, 0.06], \omega = 1.0$$

Truss Structure

Figure A.10: Optimal shape design of a truss.

The Truss structure is an optimal shape design problem. Optimizing shape design (structural optimization, or redesign) aims at reducing new product development cycle length in industries such as automotive, marine, and aerospace. Here, a truss structure design system is given in Figure A.10 as a sample shape design problem. The objective is to minimize the volume or weight of the truss structure depending on the location of node D, i.e.:

$$\min f(\mathbf{x}) = \sum_{i=1}^4 A_i \cdot L_i = \frac{1}{\sigma_0} \sum_{i=1}^4 |N_i(\mathbf{x})| \cdot \mathbf{L}_i.$$

where i is the index of rods, $\mathbf{x} = (x_w, y_w)$, $N_i(\mathbf{x})$ are axial forces, A_i is cross sectional area of rods, L_i is rod length and σ_0 is an allowable stress.

Parameters are assumed to be as follows (Pownuk, 2000): $L = 1$ m, $P = 10$ kN, allowable stress $\sigma_0 = 190$ MPa.

Weeds and Palmer 4

Weeds and Palmer 4 (CUTer) are data fitting problems where a given data set is fit into a nonlinear function. In Weeds, a logistic growth function is used, and the aim is to identify the optimal function parameters $\mathbf{x} = \{a, b, c\}$ that provide the best data fit. Palmer 4 is a 4D data fitting application and the objective is to find the optimal coefficients $\mathbf{x} = \{a, b, c, d\}$.

Weeds:

$$\min f(\mathbf{x}) = \sum_i \left(y_i - \frac{a}{1 + \exp(ci - b)} \right)^2$$

where $a = [-100, 100]$, $b = [-100, 100]$ and $c = [1.0, 3.0]$, $\mathbf{y}$ is the vector of input data to be fitted.

Palmer 4:

$$\min f(\mathbf{x}) = \sum_i \left(y_i - az_i^2 - \frac{b}{c + z_i^2/d} \right)^2$$

where $a = [0, 20]$, $b = c = d = [0, 1]$, $\mathbf{y}$ and $\mathbf{z}$ are known output and input data to be fitted, respectively.

Pressure Vessel

Cylindrical pressure vessels (Krikanov, 2000) are widely used for commercial and aerospace applications. The use of composite materials improves the performance of vessels and leads to a significant amount of material and weight savings.

Figure A.11: Cylindrical pressure vessel.

Consider a laminated cylindrical pressure vessel loaded with internal pressure p as shown in Figure A.11. We assume the vessel consists of two angle-ply layers. The first layer has thickness t_1 and is formed by symmetric filament winding under angle $\pm\varphi_1$ which is known, because the preassigned radius of the polar opening should satisfy geodetic winding conditions.

The second layer is an arbitrary angle-ply laminate characterized with thickness t_2 and angle $\pm\varphi_2$. The design parameters for the vessel shown in Figure A.11 are t_1 , t_2 and φ_2 . The objective function is to minimize the mass of the cylindrical part of the vessel, G , while satisfying the stress and strain constraints. Assume that $g_i = t_i \rho_i$, $a_i = \sigma_i / \rho_i$, $K_i = E_i / \rho_i$; here subscript i indicates the i th layer, g is the mass per unit area, ρ the density, σ the stress, a and K are normalized stress and modulus. We also assume that $\bar{a}$ is the ultimate normalized stress, $\bar{\varepsilon}$ the ultimate strain. Then, the corresponding minimization problem can be written as:

$$\min g_1 + g_2$$

subject to

$$pR/2 = a_1 g_1 \cos^2 \varphi_1 + a_2 g_2 \cos^2 \varphi_2$$

$$pR = a_1 g_1 \sin^2 \varphi_1 + a_2 g_2 \sin^2 \varphi_2$$

$$a_1/K_1 = \varepsilon_x \cos^2 \varphi_1 + \varepsilon_y \sin^2 \varphi_1$$

$$a_2/K_2 = \varepsilon_x \cos^2 \varphi_2 + \varepsilon_y \sin^2 \varphi_2$$

$$a_1 \leq a_1^*, \quad 0 < a_2 \leq \bar{a}_2$$

$$\varepsilon_x \leq \bar{\varepsilon}_x, \quad \varepsilon_y \leq \bar{\varepsilon}_y$$

where

$$a_1^* = \bar{a}_1(3 \sin^2 \varphi_2 - 2) \cos \varphi_1 / (\sin^3 \varphi_2 \sqrt{\sin \varphi_2^2 - \sin \varphi_1^2})$$

From the first four equations, a_1 , a_2 , ε_x and ε_y can be represented in terms of g_1 , g_2 and φ_2 , and substituted into the objective, leaving the last four stress and strain bound constraints to be incorporated into the objective with a penalty factor as indicated below.

$$\min g_1 + g_2 + \sum r \max(v_i - \bar{v}_i, 0)$$

where, v_i represent a_i and ε_i , and $\bar{v}_i$ are the corresponding ultimate normalized stress and strain, r is the penalty coefficient.

Protein Folding

Protein folding problem (Stillinger et al., 1993) involves two amino acids “A” and “B” that are linked together by unit length bonds forming two dimensional linear polymers. $n - 2$ angles of bend are to be optimized at each of the nonterminal residues. Here, $n = 5$ and we solve 3D protein folding

problems. In this model, it is assumed that two kinds of interactions compose the intramolecular potential energy, V_1 , backbone bend potentials, and V_2 , non-bonded interactions with species dependent Lennard-Jones (12,6) form. Residue species along the backbone are encoded by a set of binary variables, $\xi_1, \dots, \xi_n$. If $\xi_i = 1$, i^{th} residue is A , and if it is -1 , residue is B . Below the intramolecular potential-energy function is expressed generally for an n -mer. Here, r_{ij} is distance, and coefficient $C(\xi_i, \xi_j)$ is 1 for an AA pair, $1/2$ for a BB pair and $-1/2$ for an AB pair. In the experiments, all possible sequences of the $n = 5$ problem are solved, but only the ones found to produce different results under the portfolio of tested methods are reported (8 problems).

$$f(\theta) = \sum_{i=2}^{n-1} V_1(\theta_i) + \sum_{i=1}^{n-2} \sum_{j=i+2}^n V_2(r_{ij}, \xi_i, \xi_j)$$

where,

$$-\pi \leq \theta_i \leq \pi$$

$$r_{ij} = \left\{ \left[\sum_{k=i+1}^{j-1} \cos \left(\sum_{l=i+1}^k \theta_l \right) \right]^2 + \left[\sum_{k=i+1}^{j-1} \sin \left(\sum_{l=i+1}^k \theta_l \right) \right]^2 \right\}^{1/2}$$

$$V_1(\theta_i) = (1 - \cos \theta_i)/4$$

$$V_2(r_{ij}, \xi_i, \xi_j) = 4 [r_{ij}^{-12} - C(\xi_i, \xi_j)r_{ij}^{-6}]$$

$$C(\xi_i, \xi_j) = (1 + \xi_i + \xi_j + 5\xi_i\xi_j)/8$$

Appendix B

Test Problems for Constrained Optimization

Most of the test problems for constrained optimization are taken from Floudas et al. (1999), others are formed by adding constraints to the problems used for unconstrained optimization. Problems presented here are rewritten into normal form, that is, variables are expressed as x rather than the original form presented in the book.

B.1 First Batch of Problems

Test Problem 1

Section 8.2.7 in Floudas et al. (1999), $\mathbb{R}^5$, 3 constraints.

Objective function:

$$\max f(\mathbf{x}) = -(x_1 - 1)^2 - (x_1 - x_2)^2 - (x_2 - x_3)^3 - (x_3 - x_4)^4 - (x_4 - x_5)^4$$

Constraints:

$$x_1 + x_2^2 + x_3^3 - 3\sqrt{2} - 2 = 0$$

$$x_2 - x_3^2 + x_4 - 2\sqrt{2} + 2 = 0$$

$$x_1x_5 - 2 = 0$$

Variable bounds:

$$x_i = [-5, 5], \quad i = 1, \dots, 5.$$

Test Problem 2

Section 8.2.8 in Floudas et al. (1999), $\mathbb{R}^6$, 5 constraints.

Objective function:

$$\max f(\mathbf{x}) = x_4$$

Constraints:

$$x_1(1 + 0.09755988x_5) - 1 = 0$$

$$x_2(1 + 0.09658428x_6) - x_1 = 0$$

$$x_3(1 + 0.0391908x_5) + x_1 - 1 = 0$$

$$x_4(1 + 0.9x_6) - x_3 + x_2 - x_1 = 0$$

$$\sqrt{x_5} + \sqrt{x_6} - 4 \leq 0$$

Variable bounds:

$$x_i = [0, 1], \quad i = 1, \dots, 4,$$

$$x_j = [0.00001, 16], \quad j = 5, 6.$$

Test Problem 3

Section 3.3 in Floudas et al. (1999), $\mathbb{R}^5$, 6 constraints.

Objective function:

$$\begin{aligned} \max f(\mathbf{x}) = & 40792.141 - 37.293239x_1 - 0.8356891x_1x_5 - \\ & 5.3578547x_3^2 \end{aligned}$$

Constraints:

$$(-0.0022053x_3 + 0.0056858x_2)x_5 + 0.0006262x_1x_4 - 6.665593 \leq 0$$

$$(0.0022053x_3 - 0.0056858x_2)x_5 - 0.0006262x_1x_4 - 85.334407 \leq 0$$

$$0.0071317x_2x_5 + 0.0021813x_3^2 + 0.00299955x_1x_2 - 29.48751 \leq 0$$

$$-0.0071317x_2x_5 - 0.0021813x_3^2 - 0.00299955x_1x_2 + 9.48751 \leq 0$$

$$0.0047026x_3x_5 + 0.0019085x_3x_4 + 0.0012547x_1x_3 - 15.699039 \leq 0$$

$$(-0.0047026x_5 - 0.0019085x_4 - 0.0012547x_1)x_3 + 10.699039 \leq 0$$

Variable bounds:

$$x_1 = [78, 102], \quad x_2 = [33, 45],$$

$$x_3 = [27, 45], \quad x_4 = [27, 45],$$

$$x_5 = [27, 45].$$

Test Problem 4

Section 12.2.6 in Floudas et al. (1999), $\mathbb{R}^2$, 3 constraints.

Objective function:

$$\max f(\mathbf{x}) = 5x_1 - 3x_2$$

Constraints:

$$2x_2^2 - 2\sqrt{x_2} - 2\sqrt{x_1}x_2^2 + 11x_2 + 8x_1 - 39 \leq 0$$

$$x_1 - x_2 - 3 \leq 0$$

$$2x_2 + 3x_1 - 24 \leq 0$$

Variable bounds:

$$x_1 = [1, 10], \quad x_2 = [1, 6].$$

Test Problem 5

Spike, $\mathbb{R}^2$, 3 constraints.

Objective function:

$$\max f(\mathbf{x}) = 21.5 + x_1 \sin(4\pi x_1) + x_2 \sin(20\pi x_2)$$

Constraints:

$$17.35 - x_1 \sin(4\pi x_1) - x_2 \sin(20\pi x_2) \leq 0$$

$$(x_1 - 12.1)^2 + (x_2 - 4.1)^2 - 225 \leq 0$$

$$(x_1 + 3)^2 + (x_2 - 5.8)^2 - 225 \leq 0$$

Variable bounds:

$$x_1 = [-3, 12.1], \quad x_2 = [4.1, 5.8].$$

Test Problem 6

Section 12.2.1 in Floudas et al. (1999), $\mathbb{R}^5$, 5 constraints.

Objective function:

$$\max f(\mathbf{x}) = -2x_1 - 3x_2 - 1.5x_3 - 2x_4 + 0.5x_5$$

Constraints:

$$x_1^2 + x_3 - 1.25 = 0$$

$$x_2\sqrt{x_2} + 1.5x_4 - 3 = 0$$

$$x_1 + x_3 - 1.6 \leq 0$$

$$1.333x_2 + x_4 - 3 \leq 0$$

$$x_5 - x_3 - x_4 \leq 0$$

Variable bounds:

$$x_1 = [0, 2], \quad x_2 = [0.0001, 2],$$

$$x_3 = [0, 1], \quad x_4 = [0, 1],$$

$$x_5 = [0, 1].$$

Test Problem 7

Section 12.2.2 in Floudas et al. (1999), $\mathbb{R}^3$, 3 constraints.

Objective function:

$$\max f(\mathbf{x}) = 0.7x_3 - 5(x_1 - 0.5)^2 - 0.8$$

Constraints:

$$-\exp(x_1 - 0.2) - x_2 \leq 0$$

$$x_2 + 1.1x_3 - 1 \leq 0$$

$$x_1 - 1.2x_3 - 0.2 \leq 0$$

Variable bounds:

$$x_1 = [0.2, 1], \quad x_2 = [-2.22554, -1], \quad x_3 = [0, 1].$$

Test Problem 8

Section 7.2.5 in Floudas et al. (1999), $\mathbb{R}^5$, 6 constraints.

Objective function:

$$\max f(\mathbf{x}) = -37.2392x_1 - 0.8357x_1x_5 - 5.3578x_3^2$$

Constraints:

$$0.00002584x_3x_5 - 0.00006663x_2x_5 - 0.0000734x_1x_4 - 1 \leq 0$$

$$0.000853007x_2x_5 + 0.00009395x_1x_4 - 0.00033085x_3x_5 - 1 \leq 0$$

$$1330.3294/(x_2x_5) - 0.42x_1/x_5 - 0.30586x_3^2/(x_2x_5) - 1 \leq 0$$

$$0.00024186(x_2 + x_5) + 0.00010159x_1x_2 + 0.00007379x_3^2 - 1 \leq 0$$

$$2275.1327/(x_3x_5) - 0.2668x_1/x_5 - 0.40584x_4/x_5 - 1 \leq 0$$

$$0.00029955x_3x_5 + 0.00007992x_1x_3 + 0.00012157x_3x_4 - 1 \leq 0$$

Variable bounds:

$$x_1 = [78, 102], \quad x_2 = [33, 45], \quad x_3 = [27, 45],$$

$$x_4 = [27, 45], \quad x_5 = [27, 45].$$

Test Problem 9

Rastrigin(4), $\mathbb{R}^4$, 2 constraints.

Objective function:

$$\max f(\mathbf{x}) = \sum_{i=1}^4 [10 \cos(2\pi x_i) - x_i^2] - 40$$

Constraints:

$$\sum_{i=1}^4 x_i^2 - 1 = 0$$

$$\sum_{i=1}^4 x_i - 4 \leq 0$$

Variable bounds:

$$x_i = [-3.12, 5.12], \quad i = 1, \dots, 4$$

Test Problem 10

Crescent, $\mathbb{R}^2$, 3 constraints.

Objective function:

$$\max f(\mathbf{x}) = -\max\{x_1^2 + (x_2 - 1)^2 + x_2 - 1, -x_1^2 - (x_2 - 1)^2 + x_2 + 1\}$$

Constraints:

$$x_1 + x_2 - 1 \leq 0$$

$$x_2 - 0.01 \leq 0$$

$$(x_1 - 1.5)^2 + (x_2 + 0.5)^2 - 4 \leq 0$$

Variable bounds:

$$x_1 = [-1.5, 1.5], \quad x_2 = [-0.5, 2.5].$$

B.2 Second Batch of Problems

Test Problem 11

Section 5.4.2 in Floudas et al. (1999), $\mathbb{R}^8$, 6 constraints.

Objective function:

$$\max f(\mathbf{x}) = -(x_1 + x_2 + x_3)$$

Constraints:

$$x_4 + x_6 - 400 \leq 0$$

$$x_5 - x_4 + x_7 - 300 \leq 0$$

$$x_8 - x_5 - 100 \leq 0$$

$$x_1(1 - x_6) + (2500x_4 - 250000)/3 \leq 0$$

$$x_2(x_4 - x_7) - 1250(x_4 - x_5) \leq 0$$

$$x_3(x_5 - x_8) - 2500x_5 - 1250000 \leq 0$$

Variable bounds:

$$x_1 = [100, 10000], \quad x_2 = [1000, 10000], \quad x_3 = [1000, 10000],$$

$$x_i = [10, 1000], \quad i = 4, \dots, 8$$

Test Problem 12

Section 7.2.2 in Floudas et al. (1999), $\mathbb{R}^6$, 5 constraints.

Objective function:

$$\max f(\mathbf{x}) = x_4$$

Constraints:

$$x_1 + 0.09755988x_1x_5 - 1 = 0$$

$$x_2 - x_1 + 0.990.09755988x_2x_6 = 0$$

$$x_3 + x_1 + 0.0391908x_3x_5 - 1 = 0$$

$$x_4 - x_3 + x_2 - x_1 + 0.9x_4x_6 = 0$$

$$\sqrt{x_5} + \sqrt{x_6} - 4 \leq 0$$

Variable bounds:

$$\begin{aligned} x_1 &= [0, 1], & x_2 &= [0, 1], \\ x_3 &= [0, 1], & x_4 &= [0, 1], \\ x_5 &= [0.00001, 16], & x_6 &= [0.00001, 16]. \end{aligned}$$

Test Problem 13

Section 3.4 in Floudas et al. (1999), $\mathbb{R}^6$, 6 constraints.

Objective function:

$$\begin{aligned} \max f(\mathbf{x}) &= 25(x_1 - 2)^2 + (x_2 - 2)^2 + (x_3 - 1)^2 + \\ &\quad (x_4 - 4)^2 + (x_5 - 1)^2 + (x_6 - 4)^2 \end{aligned}$$

Constraints:

$$4 - (x_3 - 3)^2 - x_4 \leq 0$$

$$4 - (x_5 - 3)^2 - x_6 \leq 0$$

$$x_1 - 3x_2 - 2 \leq 0$$

$$x_2 - x_1 - 2 \leq 0$$

$$x_1 + x_2 - 6 \leq 0$$

$$2 - x_1 - x_2 \leq 0$$

Variable bounds:

$$x_i = [0, 6], \quad i = 1, \dots, 6.$$

Test Problem 14

Rastrigin(8), $\mathbb{R}^8$, 2 constraints.

Objective function:

$$\max f(\mathbf{x}) = \sum_{i=1}^8 [10 \cos(2\pi x_i) - x_i^2] - 80$$

Constraints:

$$\sum_{i=1}^8 x_i^2 - 1 = 0$$

$$\sum_{i=1}^8 x_i - 8 \leq 0$$

Variable bounds:

$$x_i = [-3.12, 5.12], \quad i = 1, \dots, 8$$

Test Problem 15

Rastrigin(10), $\mathbb{R}^{10}$, 2 constraints.

Objective function:

$$\max f(\mathbf{x}) = \sum_{i=1}^{10} [10 \cos(2\pi x_i) - x_i^2] - 100$$

Constraints:

$$\sum_{i=1}^{10} x_i^2 - 1 = 0$$

$$\sum_{i=1}^{10} x_i - 10 \leq 0$$

Variable bounds:

$$x_i = [-3.12, 5.12], \quad i = 1, \dots, 10$$

Test Problem 16

CUTEr, $\mathbb{R}^{20}$, 14 constraints.

Objective function:

$$\begin{aligned} \max f(\mathbf{x}) = & -1(0.01x_2 + 33.333x_3 + 100.0x_4 + \\ & 0.01x_5 + 33.343x_6 + 100.01x_7 + 33.333x_8 + \\ & 133.33x_9 + 100x_{10}) \end{aligned}$$

Constraints:

$$\begin{aligned} & 0.00841x_{12} - 0.08406x_{13} - 0.20667x_{14} + 20.658x_{16} + \\ & 1.07x_{18} - 10.5x_{19} = 0 \\ & 0.00629x_{11} - 0.20555x_{12} - 4.1106x_{13} + 101.04x_{15} + \\ & 505.1x_{16} - 256.72x_{19} = 0 \\ & -0.00257x_{11} + 0.25221x_{12} - 6.2x_{14} + 1.09x_{17} = 0 \\ & -0.0000025742x_{11} + 0.00252x_{13} - 0.61975x_{16} + 1.03x_{20} = 0 \\ & 0.002x_4 - 1.02x_{10} + 100x_{16} + 0.03x_{19} + 0.01x_{20} = 0 \\ & 0.02x_3 - 1.06x_8 + 100x_{14} + 0.03x_{17} + 0.01x_{19} = 0 \\ & 0.001x_3 + 0.01x_4 - 1.04x_9 + 100x_{15} + 0.03x_{18} + 0.01x_{19} = 0 \end{aligned}$$

B.2 Second Batch of Problems

$$0.001x_2 + 0.489x_4 - 101.03x_7 + 100x_{13} + 0.03x_{15} + 0.01x_{16} = 0$$

$$0.01x_2 + 0.489x_3 - 101.03x_6 + 100x_{12} + 0.03x_{14} + 0.01x_{15} = 0$$

$$0.978x_2 - 201x_5 + 100x_{11} + 0.03x_{12} + 0.01x_{13} = 0$$

$$0.00066667x_1 - 1.01x_4 + 200x_7 + 0.06x_9 + 0.02x_{10} = 0$$

$$0.0066667x_1 - 1.03x_3 + 200x_6 + 0.06x_8 + 0.02x_9 = 0$$

$$0.326x_1 - 101x_2 + 200x_5 + 0.06x_6 + 0.02x_7 = 0$$

$$-0.70785 + x_1 + 2x_2 + 2x_3 + 2x_4 + x_5 + 2x_6 + 2x_7 +$$

$$x_8 + 2x_9 + x_{10} = 0$$

Variable bounds:

$$x_i = [0, 1], \quad i = 1, \dots, 20$$