Parallel Competing Algorithms in Global Optimization

by
H.P.J. Bolton

A dissertation submitted in partial fulfilment
of the requirements for the degree of

Master of Engineering

in the Department of Mechanical and Aeronautical Engineering,
University of Pretoria

December 2000

Supervisor:
Dr. Albert A. Groenwold

Abstract

Title:	Parallel Competing Algorithms in Global Optimization
Author:	Hermanus Petrus Johannes Bolton
Degree:	M.Eng (Mechanical)
Department:	Mechanical Engineering
Supervisor:	Prof. Albert A. Groenwold
Keywords:	Global Optimization, Global Stopping Rule, Parallel Competing Algorithms, Slope Stability.

Specialized techniques are needed to solve global optimization problems, due to the existence of multiple local optima or numerical noise in the objective function. The complexity of the problem is aggravated when discontinuities and constraints are present, or when evaluation of the objective function is computationally expensive. The global (minimization) programming problem is defined as finding the variable set for which the objective function obtains not only a local minimum, but also the smallest value, the global minimum. From a mathematical point of view, the global programming problem is essentially unsolvable, due to a lack of mathematical conditions characterizing the global optimum. In this study, the unconstrained global programming problem is addressed using a number of novel heuristic approaches.

Firstly, a probabilistic *global stopping criterion* is presented for multi-start algorithms. This rule, denoted the unified Bayesian stopping criterion, is based on the single mild assumption that the probability of convergence to the global minimum is comparable to the probability of convergence to any other local minimum. This rule was previously presented for use in combination with a specific global optimization algorithm, and is now shown to be effective when used in a general multi-start approach. The suitability of the unified Bayesian stopping criterion is demonstrated for a number of algorithms using standard test functions.

Secondly, multi-start global optimization algorithms based on *multiple local searches*, combined with the unified Bayesian stopping criterion, are presented. Numerical results reveal that these simple multi-start algorithms outperform a number of leading contenders.

Thirdly, parallelization of the sequential multi-start algorithms is shown to effectively reduce the apparent computational time associated with solving expensive global programming

problems.

Fourthly, two algorithms simulating *natural phenomena* are implemented, namely the relatively new particle swarm optimization method and the well known genetic algorithm. For the current implementations, numerical results indicate that the computational effort associated with these methods is comparable.

Fifthly, the observation that no single global optimization algorithm can consistently outperform any other algorithm when a large set of problems is considered, leads to the development of a *parallel competing algorithm* infrastructure. In this infrastructure different algorithms, ranging from deterministic to stochastic, compete simultaneously for a contribution to the unified Bayesian global stopping criterion. This is an important step towards facilitating an infrastructure that is suitable for a range of problems in different classes.

In the **sixth** place, the constrained global programming problems is addressed using constrained algorithms in the parallel competing algorithm infrastructure.

The developed methods are extensively tested using standard test functions, for both serial and parallel implementations. An optimization procedure is also presented to solve the *slope stability* problem faced in civil engineering. This new procedure determines the factor of safety of slopes using a global optimization approach.

Opsomming

Titel:	Parallelle Kompeterende Algoritmes in Globale Optimering
Outeur:	Hermanus Petrus Johannes Bolton
Graad:	M.Ing (Meganies)
Departement:	Meganiese Ingenieurswese
Studieleier:	Prof. Albert A. Groenwold
Sleutelwoorde:	Globale Optimering, Globale Termineringsreël, Parallelle Kompeterende Algoritmes, Hellingstabiliteit.

Gespesialiseerde tegnieke word benodig vir die oplos van globale programmeringsprobleme, vanweë die teenwoordigheid van lokale minima of numeriese geraas in die doelfunksie. Die probleem se kompleksiteit word vererger deur diskontuiniteite en begrensings in die doelfunksie en wanneer die evaluering van die doelfunksie berekeningsgewys duur is. Die globale (minimerings) programmeringsprobleem impliseer die bepaling van die stel veranderlikes waar die doelfunksie nie net 'n lokale minimum bereik nie, maar ook die kleinste waarde, die globale minimum. Vanuit 'n wiskundige oogpunt is die globale programmeringsprobleem nie oplosbaar nie, vanweë die gebrek aan wiskundige voorwaardes wat die globale minimum beskryf. In hierdie studie word die globale optimeringsprobleem aangespreek deur 'n paar nuwe heuristiese benaderings.

Eerstens word 'n *globale termineringsreël* vir multi-begin algoritmes voorgestel. Hierdie reël, genoem die universele Bayesiaanse termineringsreël, is gebaseer op die gematigde aanname dat die waarskynlikheid vir konvergensie na die globale minimum vergelykbaar is met die waarskynlikheid van konvergensie na enige ander lokale minimum. Hierdie reël was voorheen voorgestel vir die gebruik saam met 'n spesifieke globale optimerings algoritme, en word nou aangetoon as effektief vir 'n algemene multi-begin benadering. Die reël se toepaslikheid word vir 'n verskeidenheid van algoritmes met 'n stel standaard toetsfunksies aangetoon.

Tweedens word multi-begin globale optimeringsalgoritmes voorgestel, wat gebaseer is op *veelvuldige lokale soekprosedures*, gekombineerd met die universele Bayesiaanse termineringsreël. Numeriese resultate toon dat hierdie eenvoudige benadering heelwat bekende algoritmes oortref.

Derdens word aangetoon dat parallelisering van multi-begin algoritmes die berekeningstyd gepaardgaande met die oplossing van duur globale optimeringsprobleme effektief verminder.

Vierdens is die relatiewe nuwe deeltjie swerm ('particle swarm') optimeringsalgoritme en 'n genetiese algoritme geimplimenteer. Dié twee algoritmes is gebaseer op *natuurlike verskynsels* en numeriese resultate toon dat die metodes vergelykbaar presteer vir die huidige implementerings.

Vyfdens lei die waarneming dat geen globale optimeringsalgoritme deurentyd beter as ander algoritmes kan presteer nie, tot die ontwikkeling van die *parallele kompeterende algoritme* infrastruktuur. Verskillende algoritmes, van deterministiese tot stogastiese metodes, kompeteer gelyktydig vir 'n bydrae tot die universele Bayesiaanse termineringsreël in die infrastruktuur. Hierdie is 'n belangrike stap in die fasilitering van 'n infrastruktuur wat toepaslik is vir 'n wye spektrum van probleme vanuit verskillende klasse.

In die **sesde** plek word die begrensde globale optimeringsprobleem aangespreek deur die implementering van begrense algoritmes in die parallele kompeterende algoritme infrastruktuur.

Die ontwikkelde metodes is breedvoerig met behulp van 'n stel standaard toetsfunksies getoets, vir serie sowel as parallele implementerings. 'n Optimerings prosedure is ook ontwikkel vir die *hellingstabiliteitsprobleem* in siviele ingenieurswese. Hierdie nuwe prosedure bepaal die veiligheidsfaktor vir grondhellings deur middel van 'n globale optimerings benadering.

Acknowledgments

I would like to express my sincere gratitude towards the following persons:

- Dr. A.A. Groenwold, my supervisor, for his guidance and support throughout this study. This research would have been impossible without him.
- To both my parents for their unfailing support, and for granting me the opportunity to study and financial backing.
- Dr. G. Heymann for his contribution regarding the geotechnical aspect of the slope stability analysis.
- My fellow students for their assistance with LINUX.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Numerical techniques are frequently used in science, economics and engineering to compute globally optimal solutions to practical optimization problems. Global optimization problems are extraordinarily diverse and specialized techniques are needed to solve these problems due to:

- the existence of multiple local optima,
- numerical noise,
- the presence of discontinuities,
- the presence of constraint functions,
- computationally expensive functions, and
- a large number of design variables.

The aim of global optimization is to find the solution in a design space D for which the objective function $f(\boldsymbol{x})$ obtains not only a local minimum, but its smallest value, the global minimum. More formally, the unconstrained minimization problem is expressed as follows: Consider the unconstrained (or bounds constrained) mathematical programming problem represented by the following: Given a real valued objective function $f(\boldsymbol{x})$ defined on the set $\boldsymbol{x} \in D$ in $\mathbb{R}^n$, find the point $\boldsymbol{x}^*$ and the corresponding function value f^* such that

$$f^* = f(\boldsymbol{x}^*) = \min\{f(\boldsymbol{x}) | \boldsymbol{x} \in D\} \tag{1.1}$$

if $\boldsymbol{x}^*$ exists and is unique. Alternatively, find a low approximation $\hat{f}$ to f^*. If the objective function and/or the feasible domain D are non-convex, then there may be many local minima which are not optimal. From a *mathematical* point of view, Problem (1.1) is essentially unsolvable due to a lack of mathematical conditions characterizing the global optimum, as

opposed to the attainment of the local minima which is characterized by the Karush-Kuhn-Tucker conditions.

Global optimization problems fall into the class of NP-hard problems when considering the complexity of the problem. This means that the computational time required to solve the problem increases exponentially when the number of variables are increased [1].

1.1.1 Global optimization methods

Optimization algorithms aimed at solving Problem (1.1) are divided in two classes, namely *deterministic* and *stochastic*. The first class being those algorithms which implicitly search all of the function domain and thus are guaranteed to find the global optimum. In general, the algorithms within this class are forced to deal with restricted classes of functions (e.g. Lipschitz continuous functions with known Lipschitz constants). Even with these restrictions it is often computationally infeasible to apply deterministic algorithms to search for the guaranteed global optimum as the number of computations required increases exponentially with the dimension of the feasible space. To overcome the inherent difficulties of the guaranteed-accuracy algorithms, much research effort has been devoted to algorithms in which a stochastic element is introduced, this way the deterministic guarantee is relaxed into a *confidence measure*. A number of successful algorithms belong to the latter class.

A general stochastic algorithm for global optimization consists of three major steps [2]: a sampling step, an optimization step, and a check of some *global stopping criterion*.

The selection of a suitable global stopping criterion is probably the most important step in formulating global optimization algorithms. It is also the most problematic, due to the very fact that characterization of the global optimum is in general not possible. In order to solve the problem heuristics may be introduced. One would expect a successful global optimization algorithm to be neither purely heuristic nor purely mathematical, but a combination of both. Hence global algorithms should not be judged on rigorous mathematics only. Instead, algorithms and their associated global stopping criteria should ultimately be judged on performance.

1.2 Objectives

This study is focused on the development of methods which address the unconstrained global programming problem and to a lesser extend, the constrained global programming problem. In practice, it is desired to present new algorithms and stopping criteria. These methodologies should be

- cost efficient in terms of the number of function evaluations, and
- robust in providing a high probability of finding the global minimum.

Calculating the factor of safety of slopes is important in a number of civil engineering applications. These include natural slopes, earth works construction, embankments, earth dams,

etc. In recent years finite element methods have been developed for slope stability analyses [3, 4], but limiting equilibrium methods are still widely used. Limiting equilibrium methods combined with global optimization algorithms, can be used for determining the geometry of the critical failure plain and corresponding factor of safety. The study also aims at developing a global optimization procedure for determining the factor of safety of soil slopes.

1.2.1 Testing of the developed algorithms

The performance of the optimization methods must be evaluated in some way for comparison purposes. In this study, the test functions tabulated in Table 1.1 are used to evaluate the algorithms developed, implemented and/or tested. The set represents an extended Dixon-Szegö test set, and the test problems are explicitly given in Appendix A.

No.	Acronym	Name
1	G1	Griewank G1
2	G2	Griewank G2
3	GP	Goldstein-Price
4	C6	Six-hump camel back
5	SH	Shubert, Levi No. 4
6	RA	Rastrigin
7	BR	Branin
8	H3	Hartman 3
9	H6	Hartman 6
10	S5	Shekel 5
11	S7	Shekel 7
12	S10	Shekel 10

Table 1.1: The extended Dixon-Szegö test set.

When comparing different algorithms no *a priori* known information about the objective function should be used. For example, the termination of algorithms once they reach the known global optimum within a prescribed tolerance is unrealistic, since the global minimum of problems encountered in practice would not be known. This makes the comparison of different algorithms very difficult.

1.3 Thesis overview

This thesis is constructed as follows:

- A global stopping criterion suitable for general multi-start procedures is proposed in Chapter 2.

- Simple heuristic local search algorithms are presented as efficient global optimization solvers in Chapter 3.
- The fundamentals and computer implementation of a simple genetic algorithm are discussed in Chapter 4.
- In Chapter 5 the recently new particle swarm optimization algorithm is discussed.
- In Chapter 6, parallelization of multi-start algorithms is shown to be effective in reducing the time for solving expensive global programming problems.
- The parallel competing algorithm infrastructure is motivated and implemented in Chapter 7, in which different algorithms compete for a contribution to the stopping criterion presented in Chapter 2.
- The implementation of four constrained algorithms in the parallel competing algorithm infrastructure for solving constrained global programming problems is described in Chapter 8.
- In Chapter 9 a global programming approach is presented for determining the factor of safety of slopes.
- Conclusions and recommendations regarding the developed methods are presented in Chapter 10.

The Appendices include the following:

- Appendix A presents the extended Dixon-Szegö unconstrained test set.
- Appendix B presents the constrained test set.
- Appendix C presents the derivation of the unified Bayesian global stopping criterion.
- Appendix D presents the soil slope geometries of the examples used for testing the slope stability procedure. The figures of the critical failure planes found are also included.
- Appendix E presents the critical failure plane coordinates calculated with the proposed slope stability optimization procedure.
- Appendix F reflects fragments of the FORTRAN code developed during this study, namely
  - the genetic algorithm,
  - the particle swarm optimization algorithm, and
  - the parallel competing optimization algorithm.

Chapter 2

On Global Stopping Criteria

2.1 Introduction

As mentioned in Section 1.1.1, a general stochastic global optimization algorithm consists of a number of steps, one of which is the evaluation of a suitable stopping criterion. Stopping criteria may be divided into two classes, namely passive stopping conditions, in which no information obtained during the optimization process is used, and sequential stopping rules, which make use of information obtained during the optimization process [5]. In this chapter a stopping criterion suitable for algorithms in a general multi-start procedure is considered.

In any multi-start procedure it is required to determine $\tilde{f}$, i.e.

$$\tilde{f} = \min \left\{ \tilde{f}^j, \text{ over all } j \text{ to date} \right\} \tag{2.1}$$

as the approximation to the global minimum value f^*, when j searches have been performed from j starting points. A stopping criterion should be used to prevent over-sampling. In addition, it is desirable to have an indication of the probability of convergence to the global minimum f^*. A Bayesian argument seems the proper framework for the formulation of such a criterion. Previously two such criteria have been presented, respectively by Boender and Rinnooy Kan [6], and Snyman and Fatti [7].

In the following sections, these two Bayesian global stopping criteria are briefly outlined.

2.1.1 A criterion due to Boender and Rinnooy Kan

This criterion, denoted the optimal sequential Bayesian stopping rule, is based on a Bayesian estimate of the number of local minima and the relative size of each region of attraction R_k in D.

Let W be the number of minimizers found after $\tilde{n}$ different sampling points have been sampled. Boender *et al.* [6] showed that the least number of random starts to find the global minimum in a probabilistically sense is the smallest $\tilde{n}$ value that satisfies the following equation:

$$\text{integer part of } \left[\frac{W(\tilde{n}-1)}{\tilde{n}-W-2} + \frac{1}{2} \right] = W \tag{2.2}$$

While apparently effective, computational expense prohibits using this rule for functions with a large number of local minima in D, because this rule is effectively only satisfied when all the minimizers are found.

2.1.2 A criterion due to Snyman and Fatti

Snyman and Fatti [7] developed a stopping condition for their multi-start trajectory method, which gives an indication of the probability of convergence to the global minimum f^*. In deriving the stopping condition an assumption is made regarding the probability that a random starting point will converge to the global minimum in relation to the probability of convergence to any local minimum of the function. Let α_k denote the probability that a random starting point will converge to local minimum $\hat{\boldsymbol{x}}^k$. Also, the probability of convergence to the global minimum $\boldsymbol{x}^*$ is denoted α^*. The following mild assumption, which is probably true for many functions of practical interest, is now made:

$$\alpha^* \geq \alpha_k \text{ for all local minima } \hat{\boldsymbol{x}}^k. \tag{2.3}$$

Furthermore, let r be the number of starting points from which convergence to the current best minimum $\tilde{f}$ occurs after $\tilde{n}$ random searches have been started. Then, under assumption (2.3), the probability that $\tilde{f}$ is equal to f^* is given by

$$Pr\left[\tilde{f} = f^*\right] \geq q(\tilde{n}, r) = 1 - \frac{(\tilde{n} + \bar{a})!\,(2\tilde{n} + \bar{b})!}{(2\tilde{n} + \bar{a})!\,(\tilde{n} + \bar{b})!}\,, \tag{2.4}$$

with $\bar{a} = a + b - 1$, $\bar{b} = b - r - 1$, and a, b suitable parameters of the Beta distribution $\beta(a, b)$. On the basis of (2.4) the adopted *stopping rule* becomes:

$$\text{STOP when } \Pr\left[\tilde{f} = f^*\right] \geq q^*, \tag{2.5}$$

where q^* is some prescribed desired confidence level, typically chosen as 0.99 - 0.999.

A proof of (2.5) is presented in Appendix C. However, the proof is expressed in terms of the probability of convergence to a local minimum, and not in terms of the region of attraction of the local minimum[1].

Snyman and Fatti [7] present an argument that their trajectory method is expected to satisfy assumption (2.3) for many functions, although no mathematical proof of applicability of the stopping condition is available. Nevertheless, their result is quite important and is in all probability of greater importance and more applicable than hitherto realized. Henceforth the rule of Snyman and Fatti will be denoted the unified Bayesian stopping rule.

2.2 The unified Bayesian stopping rule

In the following, it will numerically be shown that the unified Bayesian stopping criterion may be used in many multi-start procedures, albeit for a restricted class of functions. It is

[1]Studying simple 1-D search trajectories, it was observed that the definition of region of attraction of a local minimum is problematic. Strictly speaking, the region of attraction can only be defined when non-discrete search trajectories (line search or other) are employed.

simply argued that the rule can be adopted in any multi-start algorithm if the function and the algorithm comply with basic assumption (2.3). The effectiveness of (2.5) can be demonstrated numerically when the stopping condition is successfully used in different algorithms for various test functions.

The algorithms used to demonstrate the general applicability of the unified Bayesian stopping rule range from local to global optimization algorithms. Numerical experiments have shown that when using clustering [8, 9] or a genetic algorithm, it is advantageous to decrease the number of sampling points (or the population size) used, and to rather restart the algorithm a number of times using different random starting points as opposed to a large number of sampling points (or large population size) for a single run.

2.2.1 Combination with local optimization algorithms

The well known BFGS [10, 11, 12] and Polak-Ribiere [13] local minimizers, and indeed any other local minimizer, can be 'converted' into a global optimization algorithm using a random multi-start procedure, combined with the unified Bayesian stopping criterion (see Chapter 3). Initially, one hesitates to denote these local solvers global optimization algorithms, even though the algorithms are started using $\tilde{n}$ different random starting points. A clustering method seems more suitable.

Nonetheless, these global optimization algorithms perform better than a number of rigorously derived algorithms. This is evident from the numerical results presented in Table 2.1. Table 2.1 reflects on the performance of the BFGS algorithm, the Polak-Ribiere (PR) algorithm and the SQSD algorithm [14] when implemented with the unified Bayesian stopping condition (with $q^* = 0.99$ and $a = b = 1$). The results using the Snyman-Fatti (SF) algorithm [7], for which the stopping condition was originally proposed, are also shown.

	BFGS			PR			SQSD			SF		
Prob.	F	N_{fe}	$r/\tilde{n}$	F	N_{fe}	$r/\tilde{n}$	F	N_{fe}	$r/\tilde{n}$	F	N_{fe}	$r/\tilde{n}$
G1	4	5607	6 / 284	3	5696	6 / 212	0	207141	6 / 5873	0	5062	6 / 37
G2	0	952	6 / 24	1	966	6 / 17	—	—	—	3	25730	6 / 33
GP	0	204	5 / 8	0	306	5 / 9	0	355	6 / 18	0	1901	6 / 35
C6	0	90	5 / 6	0	160	5 / 8	1	122	5 / 12	0	516	5 / 6
SH	0	1718	6 / 127	0	1206	6 / 51	0	1172	6 / 166	0	12440	6 / 37
RA	3	872	6 / 155	3	1896	6 / 102	2	925	6 / 156	3	10971	6 / 99
BR	0	329	4 / 4	0	614	5 / 10	0	1384	6 / 27	0	680	4 / 5
H3	0	299	5 / 8	0	275	5 / 7	0	243	5 / 8	0	1370	5 / 6
H6	0	358	5 / 7	0	445	5 / 8	0	252	5 / 8	0	2346	5 / 7
S5	1	297	5 / 14	0	457	5 / 14	0	464	6 / 16	1	1571	5 / 13
S7	0	273	5 / 13	0	482	6 / 14	0	471	6 / 17	0	1624	5 / 13
S10	0	364	6 / 17	0	503	6 / 15	0	620	6 / 24	0	1477	5 / 12

Table 2.1: Results when combining the unified Bayesian stopping criterion with different algorithms. F indicates the number of failures out of 10 independent restarts and N_{fe} the average number of function evaluations.

Ten independent runs for each test problem are performed and the number of failures F and the average number of function evaluations N_{fe} for the runs are reported in the table. Clearly, these three algorithms outperform the SF algorithm in terms of function evaluations N_{fe} and number of failures F. r represents the number of starting points from which convergence to the current best minimum $\tilde{f}$ occurs after $\tilde{n}$ random searches have been started in each independent run. (Average values are reported.) These result shows that local minimizers can be transformed into robust and cost efficient global optimization algorithms using the unified Bayesian stopping criterion.

2.2.2 Combination with global optimization algorithms

A genetic algorithm and two well known global optimization algorithms, namely the Bayesian search implementation by Mockus [15, 16] and the clustering algorithm [8, 9], are combined with the unified Bayesian stopping rule.

In all cases, the prescribed probability of convergence is also taken as $q^* = 0.99$ and $a = b = 1$. In the following, F indicates the number of times the algorithms fails to converge to the *a priori* known global optimum f^*, for 10 independent runs of the algorithms. The probability that $\tilde{f}$ is equal to f^* is given by $q(\tilde{n}, r)$.

Bayesian optimization

The implementation of a Bayesian search strategy by Mockus is described in [15]. Table 2.2 illustrates the influence of the unified Bayesian stopping criterion on this algorithm. The results entered in the columns denoted 'Mockus' represent the stand-alone algorithm, while

	Mockus		Mockus†			
Prob.	F	N_{fe}	F	N_{fe}	$r/\tilde{n}$	$q(\tilde{n}, r)$
G1	6	354	0	3710	5 / 10	0.9915
G2	9	1442	0	74929	6 / 52	0.9932
GP	9	365	0	39416	6 / 106	0.9927
C6	0	371	0	1449	4 / 4	0.9921
SH	0	373	0	1485	4 / 4	0.9921
RA	0	194	0	776	4 / 4	0.9921
BR	7	258	0	4521	6 / 18	0.9951
H3	8	165	0	2938	6 / 19	0.9950
H6	6	404	1	5646	6 / 17	0.9953
S5	6	158	0	3672	6 / 23	0.9945
S7	9	160	0	13147	6 / 82	0.9929
S10	9	164	0	18987	6 / 118	0.9926

Table 2.2: Combination of the unified Bayesian stopping rule and the algorithm presented by Mockus. The stopping rule is included in the columns denoted Mockus†.

the algorithm combined with the new stopping rule is represented by the columns denoted Mockus†.

Clearly, the combination with the stopping criterion makes the algorithm more robust, i.e. the number of times the algorithm converges to f^* increases, albeit at additional computational effort. However, the computational effort increases dramatically only for those functions for which the stand-alone algorithm performs badly.

Clustering

The clustering algorithm [8, 9] aims at finding all the local minima that are potentially global. Points are sampled within the design space D and are grouped into clusters, with each cluster containing hopefully only one promising local minima. A local search procedure, such as the BFGS algorithm, may then be used to find the local minima corresponding to each cluster. The aim of clustering is to ensure that no computational effort is wasted in the local search procedure to find a minimum that was already determined.

In Table 2.3, similar results to those for the algorithm of Mockus are presented for the clustering algorithm. For the stand-alone algorithm, the suggested value of $100n$ sampling points is used. In each case, an expectation of 10 local minima is prescribed, since the exploitation *a priori* known information about f is undesireable. The results for the combined algorithm are obtained with very optimistic settings, namely only $10n$ sampling points, and an expectation of only 2 local minima.

For the difficult Griewank problems, the combined results are superior to the results obtained with the stand-alone algorithm, both in terms of robustness, and computational effort.

	Clustering		Clustering†			
Prob.	F	N_{fe}	F	N_{fe}	$r/\tilde{n}$	$q(\tilde{n}, r)$
G1	8	1302	1	5239	6 / 37	0.9936
G2	0	11644	0	8231	5 / 7	0.9944
GP	0	985	0	971	5 / 5	0.9978
C6	0	643	0	749	4 / 4	0.9921
SH	0	1626	0	4117	5 / 6	0.9959
RA	1	2038	0	5617	5 / 10	0.9915
BR	0	683	0	708	4 / 4	0.9921
H3	0	1232	0	884	4 / 4	0.9921
H6	0	3278	0	2832	4 / 4	0.9921
S5	0	1891	0	2684	5 / 8	0.9932
S7	0	2139	0	2831	5 / 8	0.9932
S10	0	2805	0	3620	5 / 9	0.9923

Table 2.3: Combination of the unified Bayesian stopping rule and the clustering algorithm. The stopping rule is included in the columns denoted Clustering†.

Genetic algorithm

For the GA (discussed in Chapter 4) a small population size of 8 is used. Table 2.4 illustrates the robustness of the combination of the unified Bayesian stopping criterion with the GA, except for the G2 problem for which 10 failures are still recorded.

	GA		GA†			
Prob.	F	N_{fe}	F	N_{fe}	$r/\tilde{n}$	$q(\tilde{n}, r)$
G1	6	2557	0	7891	5 / 7	0.9944
G2	10	—	10	—	—	—
GP	1	2213	0	6471	5 / 6	0.9959
C6	0	1293	0	2907	4 / 4	0.9921
SH	0	3155	0	8204	5 / 5	0.9978
RA	2	1619	0	6784	5 / 8	0.9932
BR	0	1685	0	3773	4 / 4	0.9921
H3	0	1477	0	3073	4 / 4	0.9921
H6	4	2275	0	6235	5 / 6	0.9959
S5	4	3319	0	23932	6 / 17	0.9953
S7	6	3277	0	23798	6 / 17	0.9953
S10	5	3235	0	24686	6 / 18	0.9951

Table 2.4: Combination of the unified Bayesian stopping rule and the genetic algorithm discussed in Chapter 4. The stopping rule is included in the columns denoted GA†.

Discussion

Doubtless, the results obtained above with the three algorithms are not optimal, and can be improved. For instance, the results reported by Boender *et al.* for the clustering algorithm are notably more efficient, since they use less sampling points. In addition, the mathematical justification of Bayesian optimization is violated (e.g. see [16]). Eventually, the combination of the unified Bayesian stopping rule with algorithms such as clustering or Bayesian optimization justifies further investigation. Nevertheless, extremely simple and robust global optimization algorithms have been constructed using the unified Bayesian stopping criterion. The algorithms require no 'tuning', and are free of problem dependent parameters.

Based on these results the hypothesis is made that the unified Bayesian stopping rule can be combined with many algorithms, ranging from global to local, forming efficient stochastic multi-start algorithms. The effects of the parameters of the stopping rule, namely a, b and q^*, are discussed in the following sections.

2.2.3 Beta distribution parameters

The Beta distribution $\beta(a, b)$, with parameters a and b, is used as the marginal prior distribution of α, the probability that a random selected point from the domain D will convergence to the current overall minimum. In [7] the parameter values were taken as $a = b = 1$, which means the Beta distribution correspond to the uniform prior distribution, implying that α has a prior expectation of 0.5.

The effects the values a and b have on the total number of failures F and the average function evaluations N_{fe} for different values of the prescribed confidence level q^* are studied in Table 2.5. The Polak-Ribiere algorithm is used for demonstration purposes, but similar results can be generated using different algorithms. From Table 2.5 it is noted that a and b have little effect on the number of failures and function evaluations at high values of q^*. The reason is that with high values of q^*, a relatively high number of starts must be performed before the stopping rule can be satisfied and the assumed prior distribution of α becomes less prominent.

Values of $a = b = 1$ are probably too optimistic and values of $a = 1$ and $b = 5$ are used

	$a = 1; b = 1$		$a = 1; b = 5$		$a = 5; b = 1$		$a = 1; b = 10$		$a = 5; b = 5$	
q^*	F	N_{fe}	F	N_{fe}	F	N_{fe}	F	N_{fe}	F	N_{fe}
.80	264	269	178	356	765	36	128	447	460	98
.82	264	269	160	378	765	36	114	485	358	130
.84	264	269	143	416	765	36	102	530	358	130
.86	264	269	134	475	523	73	103	576	358	130
.88	214	497	112	602	523	73	87	683	319	154
.90	189	543	91	643	523	73	84	732	293	166
.91	126	574	85	650	523	73	73	763	294	177
.92	126	574	84	689	460	98	74	778	291	188
.93	126	574	88	687	460	98	70	853	270	204
.94	120	691	80	845	404	117	74	916	245	226
.95	113	774	70	875	404	117	53	963	205	253
.96	107	790	68	895	394	140	52	1022	283	186
.97	93	900	54	1081	331	165	43	1173	162	335
.98	68	1032	44	1143	208	262	46	1246	133	419
.99	49	1309	44	1415	156	402	25	1586	104	582
.999	19	2122	13	2259	49	1143	15	2463	52	1407
.9999	10	3089	5	3261	21	2102	4	3363	13	2239
.99999	2	3972	3	4085	11	2898	2	4210	6	3081

Table 2.5: The effect of a and b in the Beta distribution $\beta(a, b)$ on the number of failures F and function evaluations N_{fe} for 100 random restarts for all 12 the test problems. N_{fe} is calculated as the average number of function evaluations for the 1200 runs performed, using the Polak-Ribiere algorithm.

throughout this study. This effectively increases the minimum number of random starts before the stopping condition can be satisfied. For example: with a prescribed stopping probability of $q^* = 0.99$, the minimum number of random starts with $a = b = 1$ is 4, but with $a = 1$ and $b = 5$ this number is 7. This reduces the probability of 'quick' convergence to strong local minima. However, the values of a and b have a minor effect when the algorithm does not converge quickly.

2.2.4 Confidence level

A salient feature of the unified Bayesian stopping criterion is that the probability of finding the global minimum can easily be increased by simply increasing the value of the prescribed confidence level q^*. This is shown in Table 2.6. The results are obtained using $a = 1$ and $b = 5$ and the results of six algorithms, namely LLS1, LLS2, GLS1, and GLS2 (all based on multiple local searches, see Chapter 3), a GA (Chapter 4), and the SF algorithm [7] are presented. At a confidence level of 0.99 a relatively high number of failures occur using most algorithms. A value of 0.999, which results in a small number of failures and a reasonable number of function evaluations, seems most applicable. For values greater than 0.999 the number of failures decreased slightly, but the function evaluations increased drastically. The decreasing number of failures to converge to f^* as q^* increases, illustrates the general applicability of the unified Bayesian stopping rule.

2.3 Summary

In this chapter the general applicability of the unified Bayesian stopping rule in a multi-start procedure is demonstrated. This stopping rule is even combined with a genetic algorithm (with a small population size) into an effective optimization algorithm. Values of $a = 1$, $b = 5$ and $q^* = 0.99$ through 0.9999 are suggested as effective parameters in the unified Bayesian stopping rule.

Algorithm	Function	$q^* = .95$	$q^* = .99$	$q^* = .999$	$q^* = .9999$
LLS1	G1	34	28	7	3
	SH	3	2	0	0
	RA	39	33	8	11
LLS2	G1	32	20	8	3
	C6	1	0	0	0
	SH	1	0	0	0
	RA	31	24	4	2
	S5	5	0	1	0
GLS1	G1	35	26	7	1
	GP	1	0	0	0
	C6	1	0	0	0
	SH	2	0	0	0
	RA	38	17	8	5
GLS2	G1	37	19	10	2
	GP	1	0	0	0
	C6	1	0	0	0
	SH	3	0	0	0
	RA	29	11	9	4
	H6	1	0	0	0
	S5	3	0	0	0
SF	G1	11	3	1	1
	G2	61	38	13	11
	SH	1	0	0	0
	RA	35	17	9	2
GA	G1	3	0	0	0
	G2	100	100	100	100
	RA	3	0	0	0
	H3	1	1	0	0
	H6	13	4	0	0

Table 2.6: Number of failures of convergence to the global optimum f^* for 100 random restarts of each algorithm for all 12 the test problems. For the problems not listed here, the number of failures is 0 for all values of q^*.

Chapter 3

Multiple Local Searches In Global Optimization

3.1 Introduction

In the previous chapter the general applicability of the unified Bayesian stopping rule is demonstrated. In all probability, the simplest global optimization algorithm is the combination of multiple local searches, combined with some probabilistic stopping criterion. In this chapter such a formulation is presented, using the unified Bayesian stopping rule.

3.2 A simple global search heuristic

In accordance with the steps presented by Schoen [2], various sequential global optimization algorithms can be constructed as follows:

1. **Initialization:** Set the counter $j := 1$, and prescribe the desired confidence level q^*.
2. **Sampling steps:** Randomly generate $\boldsymbol{x}_0^j \in D$ in $\mathbb{R}^n$.
3. **Global minimization steps:** Starting at $\boldsymbol{x}_0^j$, attempt to minimize f in a global sense by some preliminary search procedure, viz. find and record some low function value $\bar{f}^j \leftrightarrow \bar{\boldsymbol{x}}^j$.
4. **Local minimization steps:** $\bar{\boldsymbol{x}}^j$ is used as the starting point for a robust gradient based convex minimization algorithm, with stopping criteria defined in terms of the Karush-Kuhn-Tucker conditions. Record the lowest function value $\tilde{f}^j \leftrightarrow \tilde{\boldsymbol{x}}^j$.
5. **Global termination:** Assess the global convergence after k searches are completed (yielding $\tilde{\boldsymbol{x}}^k, k = 1, 2, ..., j$) using (2.5). If (2.5) is satisfied, STOP, else $j := j + 1$ and goto 2.

3.3 Multiple local searches

Pure multiple local searches are obtained if step 3 is excluded, with $\bar{\boldsymbol{x}}^j = \boldsymbol{x}_0^j$. Two such simple algorithms are now constructed, namely:

1. LLS1: multiple local searches using the bound-constrained BFGS algorithm [10, 11, 12], and
2. LLS2: multiple local searches using the unconstrained Polak-Ribiere algorithm [13].

3.3.1 Line search methods

The BFGS and the Polak-Ribiere algorithms both employ explicit line searches. The steps in a general successive line search method are as follows:

1. **Initialization:** Given a starting point $\boldsymbol{x}^0$, set $i := 0$ and determine the first search direction $\boldsymbol{u}^0$.
2. **One-dimensional minimization steps:**
   (a) Set $i := i + 1$ and determine the next point $\boldsymbol{x}^i$ by minimizing $f(\boldsymbol{x}^{i-1} + \lambda \boldsymbol{u}^{i-1})$ with respect to λ. The one dimensional minimization may be performed by any method.
   (b) Evaluate $\nabla f(\boldsymbol{x}^i)$.
   (c) Test convergence criteria. If:

$$||\boldsymbol{x}^i - \boldsymbol{x}^{i-1}|| < \varepsilon_1 \quad \text{or} \tag{3.1}$$

$$||f(\boldsymbol{x}^i) - f(\boldsymbol{x}^{i-1})|| < \varepsilon_2 \quad \text{or} \tag{3.2}$$

$$||\nabla f(\boldsymbol{x}^i)|| < \varepsilon_3 \tag{3.3}$$

   then $\boldsymbol{x}^* := \boldsymbol{x}^i$ and STOP, else continue.
   (d) Determine new search direction $\boldsymbol{u}^i$ and go to 2 (a).

The descent directions $\boldsymbol{u}^i$ for the various line search methods are chosen differently. The Polak-Ribiere descent directions are:

$$\boldsymbol{u}^0 = -\nabla f(\boldsymbol{x}^0)$$

$$\boldsymbol{u}^i = -\nabla f(\boldsymbol{x}^i) + \beta^i \boldsymbol{u}^{i-1}$$

where:

$$\beta^i = \frac{(\nabla f(\boldsymbol{x}^i) - \nabla f(\boldsymbol{x}^{i-1}))^T \nabla f(\boldsymbol{x}^i)}{||\nabla f(\boldsymbol{x}^i)||^2}$$

These search directions are mutually conjugate and the Polak-Ribiere method will therefore terminate in a finite number of steps when applied to a quadratic function. For the BFGS

algorithm, which forms part of the Quasi-Newton methods, the descent directions are given by:

$$\boldsymbol{u}^i = -H^{-1}(\boldsymbol{x}^i)\nabla f(\boldsymbol{x}^i)$$

where H^{-1} denotes the inverse of the approximated Hessian matrix. In the Quasi-Newton methods approximations are made of the Hessian matrix, avoiding the problems associated with the Hessian matrix evaluation. The BFGS algorithm has previously been used as a local phase in a global optimization algorithm by Lee and Lee (in conjunction with a genetic algorithm) [17].

The two local minimizers meet the requirement of being *robust*, if not optimal for the application.

3.4 Multiple local searches with a global phase

For both LLS1 and LLS2 a global minimization phase (step 3) is provided for, and the respective algorithms are denoted GLS1 and GLS2. The global phase simulates the trajectories of a bouncing ball (the MBB algorithm [18]), which is attractive due to its simplicity. The ball's elasticity coefficient is chosen such that the ball's energy is dissipated quickly. The governing equations of the MBB algorithm are discussed in the following subsection.

3.4.1 Modified bouncing ball trajectory algorithm

The successive random sample points $\boldsymbol{x}_0^j, j = 1, 2, ...,$ from the box D generated in step 2 are used as starting points for the MBB algorithm. For *each* sample point $\boldsymbol{x}_0^j$, a sequence of *trajectory steps* $\Delta\boldsymbol{x}_i$ and associated *projection points* $\boldsymbol{x}_{i+1}$, $i = 0, 1, 2, ...,$ are computed from the successive analytical relationships (with $\boldsymbol{x}_0 := \boldsymbol{x}_0^j$ and prescribed $V_0 > 0$):

$$\Delta\boldsymbol{x}_i = \frac{V_i t_i \cos\theta_i \nabla f(\boldsymbol{x}_i)}{||\nabla f(\boldsymbol{x}_i)||} \tag{3.4}$$

where

$$\theta_i = \tan^{-1}(||\nabla f(\boldsymbol{x}_i)||) + \frac{\pi}{2} \tag{3.5}$$

$$t_i = \frac{1}{g}\left[V_i \sin\theta_i + \left\{(V_i \sin\theta_i)^2 + 2gh(\boldsymbol{x}_i)\right\}^{1/2}\right] \tag{3.6}$$

$$h(\boldsymbol{x}_i) = f(\boldsymbol{x}_i) + k \tag{3.7}$$

with k a constant chosen such that $h(\boldsymbol{x}) > 0 \; \forall \; \boldsymbol{x} \in D$, g a positive constant, and

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \Delta\boldsymbol{x}_i \tag{3.8}$$

Each step $\Delta\boldsymbol{x}_i$ represents the simulated horizontal displacement obtained by projecting a particle in a vertical gravitational field (constant g) at an elevation $h(\boldsymbol{x}_i)$, with speed V_i and at an inclination θ_i. The angle θ_i represents the angle that the outward normal $\boldsymbol{n}$ to

the hyper surface represented by $y = h(\boldsymbol{x})$ makes, at $\boldsymbol{x}_i$ in $n+1$ dimensional space, with the horizontal. The time of flight t_i is the time taken to reach the ground corresponding to $y = 0$. For the next step, $V_{i+1} = \alpha_e V_i$, $\alpha_e < 1$, with α_e the elasticity coefficient.

More formally, the minimization trajectory for *a given sample point* $\boldsymbol{x}_0^j$ and some initial prescribed speed V_0 is obtained by computing the sequence $\boldsymbol{x}_i$, $i = 0, 1, 2, ...$, as follows.

Minimization procedure *MP*

1. For given sample point $\boldsymbol{x}_0^j$, set $\boldsymbol{x}_0 := \boldsymbol{x}_0^j$ and compute trajectory step $\boldsymbol{\Delta x}_0$ according to (3.4) - (3.7) and subject to V_0; record $\boldsymbol{x}_1 := \boldsymbol{x}_0 + \boldsymbol{\Delta x}_0$, set $i := 1$ and $V_1 := \alpha V_0$ $(\alpha < 1)$

2. Compute $\boldsymbol{\Delta x}_i$ according to (3.4) - (3.7) to give $\boldsymbol{x}_{i+1} := \boldsymbol{x}_i + \boldsymbol{\Delta x}_i$, record $\boldsymbol{x}_{i+1}$ and set $V_{i+1} := \alpha V_i$

3. set $i := i + 1$ and go to 2

In the vicinity of a local minimum $\hat{\boldsymbol{x}}$ the sequence of projection points $\boldsymbol{x}_i$, $i = 0, 1, 2, ...$, constituting the search trajectory for starting point $\boldsymbol{x}_0^j$ will converge since $\boldsymbol{\Delta x}_i \to 0$ (see (3.4)). In the presence of many local minima, the probability of convergence to a relative low local minimum is increased, since the kinetic energy can only decrease for $\alpha < 1$.

Procedure *MP*, for a given j, is successfully terminated if $||\boldsymbol{\nabla} f(\boldsymbol{x}_i)|| \leq \epsilon$ for some small prescribed positive value ϵ, or when $\alpha V_i < \beta V_0$, and $\boldsymbol{x}_i$ is taken as the local minimizer $\bar{\boldsymbol{x}}^j$ with corresponding function value $\bar{f}^j := h(\bar{\boldsymbol{x}}^j) - k$. Note that $\bar{\boldsymbol{x}}^j$ does not necessarily have to be the last point of the minimization procedure, but the $\boldsymbol{x}_i$ corresponding to the lowest function value recorded during the procedure is taken as $\bar{\boldsymbol{x}}^j$. Clearly, the condition $\alpha V_i < \beta V_0$ will always occur for $0 < \beta < \alpha$ and $0 < \alpha < 1$.

3.5 Numerical results

The algorithms are tested using the extended Dixon-Szegö test set, presented in Table 1.1. Table 3.1 shows the results for algorithms LLS1 and LLS2, while the results for algorithms GLS1 and GLS2 are shown in Table 3.2. The tables show the average number of function evaluations N_{fe} and the number of failures F to converge to the global minimum, for 10 independent runs of each problem. The stopping rule parameters utilized are $a = 1$, $b = 5$ and $q^* = 0.999$. Also included are the average number of sampling points $\tilde{n}$ and the average number of times r that the lowest minimum is found, before stopping.

The results reveal that the inclusion of the MBB global phase reduces the number of function evaluations for some problems, but for others the cost increased. For LLS1, the MBB algorithm is in general beneficial.

	LLS1				LLS2			
Prob.	F	N_{fe}	$r/\tilde{n}$	$q(\tilde{n}, r)$	F	N_{fe}	$r/\tilde{n}$	$q(\tilde{n}, r)$
G1	1	12583	9 / 798	0.9990	0	4225	9 / 155	0.9990
G2	0	2333	10 / 49	0.9995	0	4139	10 / 73	0.9995
GP	0	463	10 / 17	0.9996	0	664	10 / 20	0.9996
C6	0	168	9 / 11	0.9991	0	304	9 / 13	0.9990
SH	0	2830	9 / 223	0.9990	0	1719	10 / 57	0.9995
RA	1	1562	9 / 259	0.9990	0	3684	9 / 142	0.9990
BR	0	823	9 / 10	0.9991	0	778	9 / 11	0.9991
H3	0	601	10 / 17	0.9996	0	619	9 / 16	0.9990
H6	0	664	9 / 13	0.9990	0	716	9 / 14	0.9990
S5	0	543	10 / 28	0.9995	0	556	10 / 24	0.9995
S7	0	546	10 / 24	0.9995	0	628	10 / 25	0.9995
S10	0	596	10 / 22	0.9996	0	565	10 / 22	0.9996

Table 3.1: Numerical results for the LLS1 and LLS2 algorithms.

	GLS1				GLS2			
Prob.	F	N_{fe}	$r/\tilde{n}$	$q(\tilde{n}, r)$	F	N_{fe}	$r/\tilde{n}$	$q(\tilde{n}, r)$
G1	1	3544	9 / 148	0.9990	0	11907	9 / 442	0.9990
G2	0	3398	10 / 72	0.9995	0	3313	10 / 59	0.9995
GP	0	764	10 / 24	0.9995	0	581	10 / 17	0.9996
C6	0	330	10 / 16	0.9996	0	303	10 / 15	0.9996
SH	0	1724	10 / 65	0.9995	0	2639	9 / 114	0.9990
RA	0	3010	9 / 154	0.9990	0	3235	9 / 175	0.9990
BR	0	542	9 / 9	0.9991	0	1402	10 / 22	0.9996
H3	0	711	10 / 17	0.9996	0	566	9 / 14	0.9990
H6	0	750	9 / 14	0.9990	0	715	9 / 14	0.9990
S5	0	334	10 / 23	0.9995	0	805	10 / 26	0.9995
S7	0	413	10 / 27	0.9995	0	822	10 / 24	0.9995
S10	0	387	10 / 24	0.9995	0	865	10 / 26	0.9995

Table 3.2: Numerical results for the GLS1 and GLS2 algorithms.

3.5.1 Comparison with other methods

Tables 3.3 and 3.4 reveal that the simple sequential algorithms presented herein compare very favorably with a number of leading contenders, namely the Snyman-Fatti (SF) algorithm [7], algorithm 'sigma' [19, 20], clustering [8, 9] and the algorithm presented by Mockus [15]. All the algorithms were started from different random starting points, and the reported cost is the average number of function evaluations N_{fe} for 10 independent runs of the algorithms.

The number of failures F to converge to the global optimum for the 10 independent runs are also reported.

Prob.	LLS1	LLS2	GLS1	GLS2	SF	cluster.	Mockus	'sigma'
G1	12583	4225	3544	11907	5062	1302	354	396147
G2	2333	4139	3398	3313	25730	11644	1442	828441
GP	463	664	764	581	1901	985	365	94587
C6	168	304	330	303	516	643	371	76293
SH	2830	1719	1724	2639	12440	1626	373	139087
RA	1562	3684	3010	3235	10971	2038	194	445711
BR	823	778	542	1402	680	683	258	71688
H3	601	619	711	566	1370	1232	165	103466
H6	664	716	750	715	2346	3278	404	106812
S5	543	556	334	805	1571	1891	158	234654
S7	546	628	413	822	1624	2139	160	212299
S10	596	565	387	865	1477	2805	164	330486

Table 3.3: Average number of function evaluations N_{fe} for 10 random restarts of each algorithm for the complete test set.

Prob.	LLS1	LLS2	GLS1	GLS2	SF	cluster.	Mockus
G1	1	0	1	0	0	8	6
G2	0	0	0	0	3	0	9
GP	0	0	0	0	0	0	9
C6	0	0	0	0	0	0	0
SH	0	0	0	0	0	0	0
RA	1	0	1	0	3	1	0
BR	0	0	0	0	0	0	7
H3	0	0	0	0	0	0	8
H6	0	0	0	0	0	0	6
S5	0	0	0	0	1	0	6
S7	0	0	0	0	0	0	9
S10	0	0	0	0	0	0	9

Table 3.4: Number of failures of convergence to the global optimum for 10 random restarts of some algorithms for the complete test set.

In particular, the results for two very difficult test functions, namely Griewank G1 and Griewank G2 are encouraging: Few algorithms find the solution to G2 (which has a few thousand local minima in the region of interest), in less than 20000 function evaluations. Although the number of function evaluations obtained with the algorithm of Mockus are the lowest for most of the problems, the number of failures for this algorithm are significantly more than those recorded for the other algorithms.

3.6 Summary

A number of efficient multi-start algorithms are presented for the unconstrained global programming problem. These algorithms are based on simple local searches, combined with the unified Bayesian global stopping criterion. In addition, a global phase based on the trajectories of a bouncing ball is presented. These simple algorithms outperform a number of leading contenders.

Chapter 4

Genetic Algorithm

4.1 Introduction

Many optimization methods have been formulated by simulating natural phenomena and the search for nature-like algorithms continues. The principle of survival of the fittest found in nature, resulted in species which are well adapted to their environment despite the richness of the genetic material originally contained in the specie population. This principle forms the basis for a stochastic search strategy, denoted the genetic algorithm (GA) [21].

GA's combine survival of the fittest among population members with a structured yet randomised genetic information exchange. GA's are well suited for discrete optimization problems, use no gradient-based information and they have been applied successfully in many fields, e.g. the design of truss structures [22], the slope stability problem [23] and in chemical engineering [24]. The GA can be used for discontinuous functions, while constraints are easily incorporated.

In this chapter a brief description of the genetic algorithm is presented, based on [21, 25, 26]. Some novel operators are also presented.

4.2 The genetic algorithm operators

Genetic algorithms act on a population of possible design solutions. An initial design population, constituting of e design vectors $\boldsymbol{x}$, is created by a random selection of the variables in the variable space for each design. The design vectors are improved in subsequent generations by means of the selection, crossover and mutation operators. In the following, the term design vector is replaced with 'string'.

4.2.1 Representation of design variables

The values of the variables in the strings must be represented by an unique coding scheme. For simplicity, it is possible to use floating point coding [24]. In this coding scheme the

variable is directly represented by one possible value taken from the number of unique discrete values the variable can assume. However, binary coding is powerful and is frequently used. For example, the binary string (01101) of length $l = 5$ represents the real number 22:

$$0 \cdot 2^0 + 1 \cdot 2^1 + 1 \cdot 2^2 + 0 \cdot 2^3 + 1 \cdot 2^4 = 22$$

A real value x within bounds (x_b, x_e) is represented by binary coding in the following way:

$$x = x_{bin} \cdot \frac{(x_e - x_b)}{(2^l)} + x_b \tag{4.1}$$

where

$$x_{bin} = \sum_{i=1}^{l} z_i \cdot 2^{(i-1)} \tag{4.2}$$

and z_i can be either 1 or 0 with l the binary string length.

If the objective function has several variables, then the design vector can be represented by a concatenation of the coding of each variable [26]. For example, the three dimensional design vector

$$\boldsymbol{x} = [22\ 8\ 11]$$

with corresponding binary code (01101); (00010); (11010) is represented as

$$\boldsymbol{X} = [011010001011010]$$

4.2.2 Selection

The selection operation selects e strings from the current population to form the mating pool. The strings corresponding to low objective function values have the greatest chance to be selected for mating and therefore to contribute to the next generation. From all the numerous different selection processes possible, only the expected value selection, tournament selection and ranking selection methods are briefly discussed.

Expected value selection (roulette wheel selection)

In the expected value selection [21], also known as the roulette wheel selection, the minimization problem is converted to a maximization problem by multiplying the objective function with -1. Also, the function values must be positive and therefore a constant must be added to functions with negative values. The relative fitness p_i for each design is calculated as follows:

$$p_i = \frac{f_i}{\sum_{i=1}^{e} f_i} \qquad i = 1, 2, 3, ...e \tag{4.3}$$

where f_i denotes the function value of design i. The cumulative probability space g_j is defined as:

$$g_j = \sum_{i=1}^{j} p_i \qquad j = 1, 2, 3, ...e \tag{4.4}$$

String i is selected for the mating pool if a random number v between 0 and 1 is generated and satisfies the condition: $g_{i-1} < v \leq g_i$ with $g_0 = 0$.

Tournament selection

Tournament selection simulates the process where individuals compete for mating rights in the population [24]. In the GA, e tournaments are held between a sub group of strings chosen randomly from the existing population. The design from each tournament with the lowest function value is selected for the mating pool.

Ranking methods

After ranking the strings in ascending order according to the objective function values, the relative fitness p_i of member i is expressed as

$$p_i = \frac{t_i}{\sum_{i=1}^{e} t_i} \tag{4.5}$$

where

$$t_i = 2 \cdot \frac{(e+1-i)^c}{(e^2+e)} \tag{4.6}$$

c is taken as any value between 1 and 10 (typically 1) and e is the population size. The cumulative probability space g_j is constructed using the above defined relative fitness p_i and the strings are selected as in the expected value selection.

4.2.3 Crossover

After selecting e strings for the mating pool, new designs are explored by the crossover process. Crossover allows selected individuals to trade characteristics of their designs by exchanging parts of their strings. The mating pool strings are randomly grouped into pairs and a breaking point in the strings for each pair is chosen randomly. The values at the string positions after the breaking point are interchanged between the pair and the new designs are copied to the new generation. Crossover for each pair is applied with a given probability p_c, usually between 0.6 and 1. For example, when crossover is applied at the third crossover position of the following strings

$$\boldsymbol{X}_1 = [011|0100]$$

$$\boldsymbol{X}_2 = [010|1101]$$

the strings exchange the last four bits and become:

$$\boldsymbol{X}_1 = [011|1101]$$

$$\boldsymbol{X}_2 = [010|0100]$$

If crossover for a pair is not applied, then the unchanged parents are copied into the next generation. It is also common to use more than one braking point during crossover.

4.2.4 Mutation

The mutation operation protects against complete loss of genetic diversity by randomly changing bit values in a string. For each bit in the population a random number is generated and the bit value is changed if the random number is less than the prescribed probability of mutation p_m. The changed value can randomly be selected from the possible values from the alphabet (jump mutation), or given the value of a neighbour bit (creep mutation).

4.3 Genetic algorithm principles

From the foregoing description of the GA, the question arises how a procedure with so much randomness can exploit the current information and improve on the fitness of the design vectors. The answer lies in the fundamental schemata theorem and the phenomenon of implicit parallelism. In order to explain these two principles, the concept of schemata and its characteristics must firstly be described.

4.3.1 Similarity template/schema

Sometimes similarities between strings at certain string positions exist together with their corresponding fitnesses. According to Davidor [25] a schema (over the binary alphabet) is a string of type $(a_1, a_2, ...a_l)$ where the value of a_i can be '1','0' or '$*$'. The '$*$' symbol is a 'don't care' symbol which accepts both '1' and '0'. A schema is a template that describes a sub-space of strings that match the schema at all positions where the schema is specific (specifies either '1' or '0'), and regardless of the value the string exhibit at the positions of the '$*$' symbol.

For example: the schema of length 5 $(1, *, 0, 0, *)$ describes the following string set:

$$(1,1,0,0,1)(1,0,0,0,0)(1,1,0,0,0)(1,0,0,0,1)$$

The defining length $\delta(H)$ of the schema H is the distance between the first and last specific schema positions. This is important as it defines the number of crossover sites which can disrupt the schema, unless crossover is performed between identical strings.

The order $o(H)$ of the schema H is the number of specific positions contained in the schema. Mutation disrupts a schema if any specific position is changed.

For a string of length l and k number of possible characters, a total of $(k+1)^l$ different schemata exists and for a given string the number of schema contained in the string is 2^l. For a population of e strings, the total number of schema contained in the population is less than $e \cdot 2^l$.

4.3.2 Schema reproduction

The foregoing sections stated that the different GA processes effects the number of schemata contained in the population. Goldberg [21] quantified the effects mathematically as follows:

Selection effect

Suppose the number of strings representing a particular schema present in the population at time t is $m(H,t)$ and the average fitness of the strings representing the schema is denoted by $f(H)$. The estimated number of strings representing this schema after the selection process is then estimated as:

$$m(H,t+1) = m(H,t) \cdot \frac{f(H)}{\sum f_i} \cdot e \tag{4.7}$$

Noting that the average fitness $\overline{f}$ of the strings is given by:

$$\overline{f} = \frac{\sum f_i}{e} \tag{4.8}$$

and substituting this into (4.7), the schema growth equation becomes:

$$m(H,t+1) = m(H,t) \cdot \frac{f(H)}{\overline{f}} \tag{4.9}$$

Let the average represented fitness f of the schema be an amount $c \cdot \overline{f}$ above of below the strings average $\overline{f}$. The value of c is greater than zero when the average represented fitness is greater than $\overline{f}$ and is less than zero if otherwise. Equation (4.9) becomes:

$$m(H,t+1) = (1+c) \cdot m(H,t) \tag{4.10}$$

and after a number of selection processes:

$$m(H,t) = (1+c)^t \cdot m(H,0) \tag{4.11}$$

(4.11) states that the number of strings representing a particular schema increases exponentially if the average fitness $f(H)$ of the strings representing the schema is above the average fitness $\overline{f}$ of the population.

Crossover effect

Crossover exchanges parts of strings between members and disrupts the schemata contained in them. There are $(l-1)$ possible crossover positions for strings of length l and the schema represented by the string is destroyed if the crossover position falls within the defining length $\delta(H)$. The probability a schema H will be destroyed is therefore:

$$p_d = \frac{\delta(H)}{(l-1)} \tag{4.12}$$

and the probability of survival is

$$p_s = 1 - p_d \tag{4.13}$$

Since crossover is performed with a probability of p_c, the survival probability is given by

$$p_s > 1 - p_c \cdot \frac{\delta(H)}{(l-1)} \tag{4.14}$$

The inequality sign includes the possibility that crossover between similar strings occurs.

Mutation effect

The probability a specific string position being disrupted by mutation is given by p_m and the survival probability thus is $(1-p_m)$. The schema survives if all the specific positions remain unchanged and therefore the survival probability becomes $(1-p_m)^{o(H)}$. Because $p_m \ll 1$, the schema survival probability when mutation are applied is approximated by:

$$p_s = 1 - o(H)p_m \tag{4.15}$$

4.3.3 Fundamental schemata theorem

The fundamental schemata theorem is formed by combining the effects mutation, crossover and selection have on the number of strings contained in the population representing a specific schema H:

$$m(H, t+1) - m(H, t) \cdot \frac{f(H)}{\overline{f}} \left[1 - p_c \cdot \frac{\delta(H)}{(l-1)} - o(H)p_m\right] \tag{4.16}$$

In words, the schema theorem states that the number of strings representing a specific schema is increasing exponentially if the strings representing a specific schema have above average fitnesses and low defining lengths and orders.

4.3.4 Implicit parallelism

In essence, the changing population can be seen as a search through the set of schemata contained in the strings. A great deal of information regarding the fitnesses of the possible schemata with one string evaluation is attained, since each string forms part of 2^l schemata. Between 2^l and $e \cdot 2^l$ possible schema exist in the population, but it is suggested that only e^3 number of schemata are simultaneously being processed, called implicit parallelism [21].

4.4 Genetic algorithm implementation

Here, a basic GA implementation is modified in an attempt to improve the performance. The GA presented here consists of a number of independent GA runs ($\mathrm{GA}_k; k = 1, 2, ..., k_{max}$) in which the variable bounds are decreased with each run. For each GA run, the probability of mutation p_m varies linearly from p_{beg} for the first generation to p_{end} for the last generation. Formally, the successive GA algorithm is presented as follows:

1. **Initialization:** Prescribe the maximum number of generations per GA run g_{max}, the population size e, the number of binary bits representing one variable l, the variable bounds decreasing factor Δx, the finishing size of the variable bounds Δx_{end}, the convergence tolerances ε_1 and ε_2, the probability of mutation limits p_{beg} and p_{end}, the maximum number of no-improvement generations b_{max}. Set the GA run counter $k := 1$.

2. **Determine number of GA runs:** Calculate the maximum number k_{max} of GA runs as follows:

$$k_{max} = \text{integer of } \left[\frac{log(\Delta x_{max}/\Delta x_{end})}{log(\Delta x)} + 1\right] \geq 1 \tag{4.17}$$

where Δx_{max} denotes the greatest variable bound size.

3. **Genetic Algorithm:**

   (a) **Generate first population:** GA run number k starts with the random generation of the first population of strings. Set the generation number counter $g := 1$.

   (b) **Function evaluation:** Determine the variable values represented by the population of strings and evaluate $f(\boldsymbol{x})$ for all strings. Record the lowest function value $\bar{f}^g \leftrightarrow \bar{\boldsymbol{x}}^g$ for the current population.

   (c) **Selection:** Select e strings from the current population, using tournament or ranking selection.

   (d) **Crossover:** Choose pairs randomly from the selected population and exchange string parts between randomly selected crossover position for each pair.

   (e) **Mutation:** Determine the probability of mutation p_m for the current generation g :

$$p_m = p_{beg} + \frac{(p_{end} - p_{beg}) \cdot g}{g_{max}} \tag{4.18}$$

   and perform jump mutation.

   (e) **Minimum updating:** Record the lowest minimum f_{min} for GA run k and the number of generations b no improvement is made on this value: If $g = 1$ or $f_{min} - \bar{f}^g > \varepsilon_1$ set $f_{min} := \bar{f}^g$ and $b := 1$. Else if $||f_{min} - \bar{f}^g|| < \varepsilon_1$ set $b := b + 1$.

   (g) **Termination of GA run k:** If the maximum number of generations $g = g_{max}$ is reached or when the maximum number of no-improvement generations $b = b_{max}$ is reached goto 4, else continue with the next generation $g := g + 1$ and goto 3 (b).

4. **Convergence test:** Set $\tilde{f}_k := f_{min}$ with $\tilde{f}_k \leftrightarrow \tilde{\boldsymbol{x}}_k$. If the maximum number of GA runs is reached $k = k_{max}$ or when the difference in minima found by two successive GA's are negligible $||\tilde{f}_k - \tilde{f}_{k-1}|| < \varepsilon_2$, then goto 6. Else continue with 5.

5. **Changing variable bounds:** Change the upper $\boldsymbol{x}^e$ and lower $\boldsymbol{x}^b$ limits of the variables according to the following equations:

$$\boldsymbol{x}^e_{k+1} = \tilde{\boldsymbol{x}}_k + \frac{\boldsymbol{x}^e_k - \boldsymbol{x}^b_k}{2 \cdot \Delta x} \tag{4.19}$$

$$\boldsymbol{x}^b_{k+1} = \tilde{\boldsymbol{x}}_k - \frac{\boldsymbol{x}^e_k - \boldsymbol{x}^b_k}{2 \cdot \Delta x} \tag{4.20}$$

   Set $k := k + 1$ and goto 3 (a).

6. **STOP:** Take $\tilde{f}_k$ as approximation to f^*.

Numerical experiments revealed that suitable values for the parameters are g_{max}=30, e=20, l=10, $\Delta x = 3$, $\Delta x_{end} = 10^{-4}$, $\varepsilon_1 = 10^{-3}$, $\varepsilon_2 = 10^{-6}$, $p_{beg} = 0.08$, $p_{end} = 0.05$ and b_{max}=6. Ranking selection is advised with $c = 5$. Appendix F.1 presents the code for the successive GA algorithm.

4.4.1 On the selection process

The developed GA has the option between tournament selection and ranking selection (Step 3 (c)). The expected value selection was at first implemented, but is not included due to poor performance when compared to the other two methods. Groenwold et al [22] used the ranking selection with the exponent c taken as unity. Numerical experiments revealed that larger values for c improves this GA's efficiency. The selection technique employed with larger c values, chooses the best fit strings more often than the lower ranking strings. This may appear to be a weak selection technique in that much genetic information is lost. However, this GA constructs a total new population at the beginning of each inner GA run (Step 3 (a)). Consequently, a whole new genetic pool is constructed which overcomes the problem of premature convergence.

4.5 Comparison between different GA implementations

The performance of the GA implementation as outlined in Section 4.4 is compared to a basic GA implementation presented by Carroll [24]. Table 4.1 shows the results for both GA implementations when each test problem from the set listed in Table 1.1 was performed 10 times from different starting points. Clearly, the successive GA method presented in this study requires less function evaluations and finds the global minimum more often than the basic GA. The reason for the excellent performance lies in the exploitation of the most significant bit property.

4.5.1 Most significant bit property

The most significant bits are the string positions where a change of bit value effects the variable value it represents most. For example, consider the binary bit string of length 5: (1,0,0,0,1) represents a variable value of 17 since $1 \cdot 2^0 + 1 \cdot 2^4 = 17$. If the first bit is changed to zero the string represents 16, since $1 \cdot 2^4 = 16$. Changing the last bit to zero and the first bit back to 1, means that the variable changes much more and now equals 1. The most significant bits in a binary string representation are thus the ending bit positions.

A GA will quickly find the values of the significant bits corresponding to good fitness, but is slow in finding the least significant bit values. The changing bound technique as explained in Section 4.4 exploits this property of the most significant bits in that as soon as the significant bits are found and the GA struggles to find the least significant bit values, a next GA starts

	GA [24]				Successive GA			
Prob.	F	N_{fe}	$\lvert\tilde{f}_{ave} - f^*\rvert$	$\lvert\tilde{f}_{best} - f^*\rvert$	F	N_{fe}	$\lvert\tilde{f}_{ave} - f^*\rvert$	$\lvert\tilde{f}_{best} - f^*\rvert$
G1	6	4026	0.5670E-01	0.6912E-08	6	2557	0.8737E-01	0.1006E-09
G2	10	7196	0.7623E+00	0.2174E+00	10	5553	0.1001E+00	0.2917E-01
QP	4	4691	0.6250E+00	0.9167E-09	1	2213	0.8100E+01	0.4644E-08
C6	5	4056	0.6278E-04	0.4654E-07	0	1293	0.3711E-06	0.4783E-07
SH	6	4016	0.2082E+00	0.1466E-04	0	3155	0.5537E-05	0.2426E-05
RA	4	3606	0.6394E-01	0.2965E-09	2	1619	0.2422E-01	0.1055E-07
BR	8	4246	0.6954E-03	0.3048E-07	0	1685	0.1691E-06	0.9433E-08
H3	6	5601	0.5549E-03	0.1190E-06	0	1477	0.2152E-05	0.1179E-06
H6	10	6886	0.4091E-01	0.5307E-02	4	2275	0.4783E-01	0.1001E-06
S5	10	4636	0.5252E+01	0.6820E-01	4	3319	0.2756E+01	0.3276E-06
S7	10	4511	0.7075E+01	0.6129E+01	6	3277	0.4492E+01	0.2570E-06
S10	10	3771	0.5671E+01	0.3100E-02	5	3235	0.3915E+01	0.4199E-06

Table 4.1: The results for the successive GA method and a basic GA. N_{fe} denotes the average number of function evaluations, F the number of failures to converge to the global minimum, $\tilde{f}_{ave}$ the average of the minima found and $\tilde{f}_{best}$ the lowest minimum found for the 10 runs. f^* denotes the global minimum.

with a smaller variable bound size. The convergence to the optimum is thus not performed by locating the least significant bit values, but through finding the most significant bit values for small variable bound sizes. The ranking selection technique with a large c value makes it possible for the GA to distinguish better fit strings from less fit strings for small variable bound sizes.

4.6 Summary

An optimization algorithm consisting of successive GA runs in which the variable bounds are continuously decreasing, is presented in this chapter. Numerical results shows that this implementation of a GA outperforms a basic GA implementation.

Chapter 5

Particle Swarm Optimization

5.1 Introduction

Genetic algorithms (GA) [21], simulated biological growth (SBG) [27] and simulated annealing (SA) [28] are well known optimization methods simulating natural phenomena.

In recent years an efficient optimization algorithm that mimics the social behaviour of bird flocks or fish schools was developed by Kennedy and Eberhart [29], called the particle swarm optimization algorithm (PSOA). This algorithm is based on the natural phenomena that members of a school or swarm benefit from the discoveries and experience of all other members of the school or swarm. The sharing of information between individuals ensures that the school or swarm explores spaces that enhances the survival of the specie. Fourie and Groenwold [30] applied the particle swarm optimization algorithm successfully to the optimal design of structures with sizing and shape variables.

In this chapter, a simple PSOA is applied to global optimization.

5.2 Mimicking social behaviour

The PSOA uses a 'swarm' of possible designs and the governing principle behind the PSOA is that every member of the swarm remembers the best position it has passed through. Through intercommunication between members, the overall best position is determined. In the following, the term member is replaced with particle and each particle position represents the design variables $\boldsymbol{x}$. In minimization, the best position corresponds to the design vector $\boldsymbol{x}$ which attains the lowest objective function $f(\boldsymbol{x})$. The movement of each particle is determined by the overall minimum recorded by the members and each member's minimum value for $f(\boldsymbol{x})$ obtained during their search paths.

5.2.1 Particle swarm equations

Following Shi and Eberhart [31], the position and velocity of particle i is updated in the following way:

$$\boldsymbol{x}_{k+1}^i = \boldsymbol{x}_k^i + \boldsymbol{v}_{k+1}^i \tag{5.1}$$

$$\boldsymbol{v}_{k+1}^i = w\boldsymbol{v}_k^i + c_1 r_1(\boldsymbol{p}_k^i - \boldsymbol{x}_k^i) + c_2 r_2(\boldsymbol{b}_k - \boldsymbol{x}_k^i) \tag{5.2}$$

$$\boldsymbol{v}_0^i = \boldsymbol{0} \tag{5.3}$$

Here, the best position of particle i is represented as $\boldsymbol{p}_k^i$, while the best swarm position up to iteration k is recorded as $\boldsymbol{b}_k$. The term $\boldsymbol{v}_{k+1}^i$, called the velocity, represents the change in position between iterations for particle i. The attraction particle i has to the overall best position $\boldsymbol{b}_k$ is controlled by the product $c_2 r_2$, while $c_1 r_1$ controls the attraction particle i has to its own best position $\boldsymbol{p}_k^i$. c_1 and c_2 are two constants, while random numbers between 0 and 1 are generated for r_1 and r_2. Kennedy and Eberhart [29] used a value of 2 for both c_1 and c_2. Continuity of the path of particle i is improved by introducing the inertia term, $w\boldsymbol{v}_k^i$. Shi and Eberhart [31] found that values between 0.8 and 1.4 are suitable for w.

5.3 Numerical results

A simple PSOA is implemented in FORTRAN and the code is given in Appendix F.2. Numerical results using the PSOA algorithm are compared to other methods that also operate on a population of solution vectors, namely the GA implemented in this study, Clustering [8, 9] and the algorithm of Mockus [15]. The current values for the parameters in the PSOA algorithm are as follows: p=20, c_1=1.0, c_2=1.0, w=0.8, v_{max}=30.0, k_{max}=4000 and r_{max}=30. v_{max} represents the maximum step limit for any particle ($0 < ||\boldsymbol{v}_{k+1}^i|| \leq v_{max}$) and k_{max} the maximum number of time steps k. The number of time steps for which the overall best swarm position $\boldsymbol{b}_k$ does not change is recorded, and the algorithm is terminated when a value of r_{max} is reached. p represents the number of particles.

The test problems presented in Table 1.1 are used for comparison and Table 5.1 reflects the performance of the PSOA. The values reported are the average number of function evaluations N_{fe} and the number of failures F to converge to the known global minimum for 10 independent runs of each problem. Each of the algorithms excelled for some problems, although the clustering algorithm in general outperforms the other algorithms. The results of the GA and the PSOA algorithms are roughly comparable.

5.4 PSOA combined with the unified Bayesian stopping criterion

The results in Table 5.1 shows that the PSOA failed to converge to the global optima a number of times for some problems. The robustness of the PSOA can be improved by using smaller number of particles and combining the algorithm with the unified Bayesian stopping

	PSOA		GA		Cluster		Mockus	
Prob.	F	N_{fe}	F	N_{fe}	F	N_{fe}	F	N_{fe}
G1	0	1776	6	2557	8	1302	6	354
G2	10	—	10	—	0	11644	9	1442
GP	2	1610	1	2213	0	985	9	365
C6	0	1262	0	1293	0	643	0	371
SH	0	2138	0	3155	0	1626	0	373
RA	6	1390	2	1619	1	2038	0	194
BR	0	1332	0	1685	0	683	7	258
H3	0	1486	0	1477	0	1232	8	165
H6	4	2356	4	2275	0	3278	6	404
S5	7	1896	4	3319	0	1891	6	158
S7	5	2190	6	3277	0	2139	9	160
S10	6	1964	5	3235	0	2805	9	164

Table 5.1: Comparing the PSOA with other methods.

criterion. Table 5.2 shows that the function evaluations cost increased, but the number of failures decreased notably for such a combination with $p = 13$ and $r_{max} = 20$.

	PSOA			
Prob.	F	N_{fe}	$r/\tilde{n}$	$q(\tilde{n}, r)$
G1	0	10524	9 / 10	0.9991
G2	10	–	–	–
GP	0	9135	9 / 10	0.9991
C6	0	6139	9 / 9	0.9991
SH	0	12197	9 / 10	0.9991
RA	0	22804	10 / 26	0.9995
BR	0	7010	9 / 9	0.9991
H3	0	9171	9 / 10	0.9991
H6	0	41635	10 / 21	0.9996
S5	0	45004	10 / 34	0.9995
S7	0	39122	10 / 29	0.9995
S10	0	40969	10 / 30	0.9995

Table 5.2: Results for the PSOA combined with the unified Bayesian stopping rule.

5.5 Summary

A simple implementation of the PSOA shows that the algorithm can effectively be used in global optimization. Doubtless, the performance can be improved through other operators, e.g. decreasing of the inertia constant w and the maximum allowed velocity v_{max}, and by introducing craziness [30]. The combination of a PSOA (employing a reduced number of particles) with the unified Bayesian stopping criterion, forms a robust global optimization algorithm.

Chapter 6

Multiple Parallel Local Searches in Global Optimization

6.1 Introduction

The combination of a multi-start local search strategy and the unified Bayesian global stopping criterion for solving the unconstrained global optimization problem is presented in Chapter 3. In structural optimization however, each function evaluation typically involves a complete finite element or boundary element analysis. Performing structural optimization using these efficient algorithms in sequential form, may still be extremely time-consuming. The search paths for the sequential algorithms presented in Section 3.2 are completely independent of each other. Hence the sequential algorithm may easily be parallelized.

In this chapter, the unconstrained global programming problem is addressed using an efficient multi-start algorithm, in which the local searches presented in Chapter 3 are performed simultaneously on different computers for a contribution towards the unified Bayesian global stopping criterion (Chapter 2).

6.2 Parallel implementation

The freely available pvm3 [32] code for FORTRAN, running under the Linux operating system, is utilized as the software system that enables a cluster of computers to be used as a massive parallel processing virtual machine (MPPVM). Currently, the MPPVM consists of up to 128 Pentium III 450 MHz machines in an existing undergraduate computer lab.

The distributed computing model represents a master-slave configuration where the master program assigns tasks and interprets results, while the slaves compute the search paths. The workload is statically assigned, and no inter-slave communication occurs. The master program informs each slave task individually of the optimization problem parameters, and awaits individual optimization results from each slave. A slave continues with a new search path after sending results back to the master program, until the unified Bayesian global stopping criterion is satisfied.

6.2.1 A measure of computational effort

It is assumed that the parallelized algorithm will ultimately be used in problems for which the CPU requirements of evaluating the objective function is orders of magnitudes larger than the time required for message passing and algorithm internals. This is the case, for example, when every function evaluation involves a complete finite element analysis, boundary element analysis or computational fluid dynamics analysis.

Hence a somewhat unconventional measure for the cost of the parallelized algorithm is defined, which is denoted *apparent visible cost* (N_{vc}). This cost represents the maximum number of function evaluations N_{fe} recorded for any slave during the parallel optimization. The time window (in CPU seconds) associated with this number of function evaluations is denoted the *virtual CPU time.* The virtual CPU time includes the time window associated with initialization and evaluation of stopping criterion (2.5).

6.3 Numerical results

Any optimization algorithm may be used for the global and local phases of the multi-start strategy presented in Section 3.2. As an example, the GLS1 algorithm is used here.

If many computers are available, it seems sensible to place a lower limit on the number of random searches $\tilde{n}$ performed, before terminating the multi-start algorithm. This mainly improves the probability of convergence without increasing the computational cost (since the $\tilde{n}$ searches are performed in parallel).

Table 6.1 reveals the effect of parallel implementation. The results of the sequential GLS1 algorithm are compared to the parallel implementation of GLS1, using a MPPVM consisting of 32 and also 128 machines. The minimum number of sampling points $\tilde{n}$ are 20 for the 32-node cluster and 90 for the 128-node cluster, representing in each case some 70 % of the available number of nodes.

For relatively 'simple' problems (viz. problems with few design variables or few local minima in the design space), the probability of convergence to the global optimum becomes very high when the number of nodes is increased. This is illustrated by, for example, the results for the C6 problem.

Simultaneously, the total computational time, (as compared to the sequential GLS1 algorithm), decreases. For the 32-node parallel virtual machine, the virtual CPU time to evaluate all the test functions on average decreases by a factor of more or less 2 (not shown in tabulated form). This low factor is a result of the inexpensive analytical test functions and the time associated with message passing. For more difficult problems (e.g. the G1 and G2 problems), the computational effort decreases drastically. When the time associated with a single function evaluations become much larger than the time required for algorithm internals, the fraction N_{fe}/N_{vc} based on Table 6.1 may be used as a direct indication of the decrease in virtual computational time obtainable as a result of parallelization. For the G2 problem, this would imply a reduction in computational time by a factor of roughly 75 for the 128-node parallel virtual machine.

	GLS1			32-node pvm			128-node pvm		
Prob.	N_{fe}	$r/\tilde{n}$	$q(\tilde{n}, r)$	N_{vc}	$r/\tilde{n}$	$q(\tilde{n}, r)$	N_{vc}	$r/\tilde{n}$	$q(\tilde{n}, r)$
G1	3544	9 / 148	0.9990	147	9 / 162	0.9990	55	9 / 166	0.9990
G2	4996	9 / 106	0.9990	189	10 / 93	0.9995	67	10 / 97	0.9995
GP	695	10 / 22	0.9996	47	10 / 24	0.9995	49	42 / 90	1.0000
C6	350	10 / 17	0.9996	26	12 / 20	0.9999	26	53 / 90	1.0000
SH	1739	10 / 66	0.9995	71	10 / 65	0.9995	44	16 / 90	1.0000
RA	2817	9 / 144	0.9990	130	9 / 167	0.9990	45	9 / 202	0.9990
BR	472	9 / 9	0.9991	54	20 / 20	1.0000	46	89 / 90	1.0000
H3	701	10 / 17	0.9996	59	11 / 23	0.9998	54	49 / 90	1.0000
H6	683	9 / 13	0.9990	48	20 / 20	1.0000	51	89 / 90	1.0000
S5	384	10 / 27	0.9995	27	10 / 24	0.9995	28	35 / 90	1.0000
S7	363	10 / 23	0.9995	28	10 / 24	0.9995	27	40 / 90	1.0000
S10	381	10 / 24	0.9995	31	10 / 29	0.9995	28	35 / 90	1.0000

Table 6.1: Apparent visual cost N_{vc} for a 32-node parallel virtual machine and a 128-node parallel virtual machine. N_{vc} may be compared with the number of function evaluations N_{fe} of the sequential GLS1 algorithm. r represents the number of starting points from which convergence to the current best minimum $\tilde{f}$ occurs after $\tilde{n}$ random searches have been started. The probability that $\tilde{f}$ is equal to f^* is given by $q(\tilde{n}, r)$.

6.4 Summary

Parallelization of multi-start algorithms is shown to be effective in reducing the computational time associated with the solution of expensive global programming problems. For example, the parallelized GLS1 algorithm is applied to the (extremely expensive) optimal design of composite shell structures using a finite element approximation by Schutte *et al.* [33].

While the apparent computational effort is reduced, the probability of convergence to the global optimum is simultaneously increased as a result of parallelization.

Chapter 7

Parallel Competing Algorithms In Global Optimization

7.1 Introduction

It is often asked which algorithm can be considered the leading global optimization algorithm. In the literature, numerical results reveal that no single global optimization algorithm can consistently outperform all other algorithms when a large set of problems in different classes is considered. Hence, a sensible approach is to attempt to solve the unconstrained global programming problem using different algorithms simultaneously.

In this chapter, the unconstrained global programming problem is addressed using different algorithms competing in parallel in a multi-start procedure.

7.2 Motivation

As stated in the above, it is observed that no single global optimization algorithm can consistently outperform all other algorithms when large sets of problems in different classes are considered. A cursory glance at numerical results in the literature suffices to impress this observation. Nevertheless, this obvious but important observation is emphasized in the following.

In Tables 7.1 and 7.2, numerical results are presented for the well known (and difficult) Griewank function. The results for the DIRECT [34], DOT [35] and LFOPC [36, 37, 38] algorithms are taken from [39]. Note that the simple GLS1 algorithm presented in Section 3.2 by far outperforms these algorithms.

Table 7.3 presents similar results to the above for the Rastrigin function. Here, the Bayesian algorithm of Mockus is by far superior to the other algorithms evaluated.

Table 7.4, similar to Table 3.3 presented in Section 3.5.1, further emphasises that no algorithm can be considered the leading global optimization algorithm. Due to the large number of times the Bayesian search implementation by Mockus fails to converge to f^* (Table 2.2),

n	Algorithm	Successes	N_{fe} (Ave.)
5	DIRECT	87 / 100	3400
	DOT	5 / 300	10260
	LFOPC	60 / 400	9050
	GLS1	500 / 500	1034
10	DIRECT	56 / 100	11810
	DOT	96 / 500	1260
	LFOPC	136 / 500	32240
	GLS1	497 / 500	977
20	DIRECT	8 / 200	102650
	DOT	16 / 2000	19740
	LFOPC	128 / 2000	7220
	GLS1	469 / 500	476

Table 7.1: Results for the Griewank Function ($n = 5,\ 10,\ 20$, with $d = 200,\ 1000,\ 20000$ respectively). 'Successes' indicates the number of times the algorithms converged to f^*.

Algorithm	d	Successes	N_{fe} (Ave.)
DIRECT	4000	19 / 100	39480
	1000	56 / 100	11810
	200	83 / 100	6990
DOT	4000	37 / 500	3970
	1000	96 / 500	1260
	200	160 / 500	620
LFOPC	4000	25 / 500	193410
	1000	136 / 500	32240
	200	390 / 500	830
GLS1	4000	498 / 500	1289
	1000	497 / 500	977
	200	500 / 500	413

Table 7.2: Effect of variation of d in the 10-dimensional Griewank Function.

the reported results for this algorithm include the unified Bayesian stopping rule. The table also includes the results of the SQSD algorithm (explained in Section 7.3.2 to come). This algorithm is converted to a global optimization algorithm using the unified Bayesian stopping rule (with $a = 1$, $b = 5$ and $q^* = 0.999$).

n	Algorithm	Successes	N_{fe} (Ave.)
2	LLS1	9/ 10	1487
	Clustering	9/ 10	3420
	LFOPC	10/ 10	45273
	Mockus	10/ 10	251
5	LLS1	2/ 10	6548
	Clustering	0/ 10	—
	LFOPC	3/ 10	348937
	Mockus	10/ 10	482
10	LLS1	2/ 10	24281
	Clustering	0/ 10	—
	LFOPC	0/ 10	—
	Mockus	10/ 10	964
20	LLS1	0/ 10	—
	Clustering	0/ 10	—
	LFOPC	0/ 10	—
	Mockus	10/ 10	1928

Table 7.3: Results for the Rastrigin Function ($n = 2,\ 5,\ 10,\ 20$).

Problem	LLS1	LLS2	GLS1	GLS2	SF [7]	[8, 9]	[19, 20]	[15]	SQSD
G1	12583	4225	3544	11907	5062	1302	396147	3710	183604
G2	2333	4139	3398	3313	25730	11644	828441	74929	—
GP	463	664	764	581	1901	985	94587	39416	649
C6	168	304	330	303	516	643	76293	1449	275
SH	2830	1719	1724	2639	12440	1626	139087	1485	1817
RA	1562	3684	3010	3235	10971	2038	445711	776	1445
BR	823	778	542	1402	680	683	71688	4521	3045
H3	601	619	711	566	1370	1232	103466	2938	506
H6	664	716	750	715	2346	3278	106812	5646	417
S5	543	556	334	805	1571	1891	234654	3672	713
S7	546	628	413	822	1624	2139	212299	13147	926
S10	596	565	387	865	1477	2805	330486	18987	990

Table 7.4: Results using different algorithms for the extended Dixon-Szegö test set. For the problems listed, the number of function values N_{fe} for the different algorithms are reported. Clearly, no algorithm consistently outperforms the other algorithms. '—' indicates that the algorithm failed to find the global optimum f^* for 10 independent runs.

7.3 Multiple competing algorithms

Based on the foregoing, a sensible, if somewhat unconventional, approach is to attempt to solve global programming problems using a number of different algorithms simultaneously.

The results of all the different algorithms combined may then be used to study the quality of local minima found. Obviously, this approach is senseless if the different algorithms are incorporated in a sequential algorithm. On the other hand, multiple algorithms in parallel are very viable.

A complication when using different algorithms simultaneously once again relates to the selection of a global stopping criterion. However, if assumption (2.3) holds for a given algorithm and objective function, then stopping criterion (2.5) may be used. Hence, multiple algorithms are implemented in an infrastructure, which compete for a contribution towards the unified Bayesian stopping criterion.

Currently, the various unconstrained algorithms competing for a contribution to the unified Bayesian global stopping criterion (2.5) are GLS1, GLS2, LLS1, a genetic algorithm (GA) (Chapter 4), the Snyman-Fatti algorithm [7], the relatively new particle swarm optimization algorithm (PSOA) (Chapter 5), clustering [8, 9], the SQSD algorithm [14] and the Bayesian search algorithm presented by Mockus [15]. Some of these algorithms are briefly reviewed in the following.

7.3.1 Leapfrog algorithm

The Leapfrog algorithm was developed by Snyman [36, 37] and is derived by considering the motion of a particle with unit mass in a conservative force field. The objective function being minimized represents the particle's potential energy.

The method therefore requires the solution to the differential equation with initial conditions:

$$\ddot{\boldsymbol{x}}(t) = -\nabla F(\boldsymbol{x}(t)) \tag{7.1}$$

$$\boldsymbol{x}(0) = -\boldsymbol{x}_0; \quad \dot{\boldsymbol{x}}(0) = \boldsymbol{v}_0 \tag{7.2}$$

From (7.1) it follows that for time interval $[0, t]$:

$$\frac{1}{2}||\dot{\boldsymbol{x}}(t)||^2 - \frac{1}{2}||\dot{\boldsymbol{x}}(0)||^2 = F(\boldsymbol{x}(0)) - F(\boldsymbol{x}(t)) \quad \text{or} \tag{7.3}$$

$$T(t) - T(0) = F(0) - F(t) \tag{7.4}$$

(7.4) indicates that if the kinetic energy $T(t)$ increases then the objective function $F(t)$ must be decreasing. Using numerical integration the initial value problem is solved and $\dot{\boldsymbol{x}}(t)$ is monitored. The particle moves uphill when $\frac{1}{2}||\dot{\boldsymbol{x}}(t)||^2$ decreases and an interfering strategy is then applied to extract the energy to increase the likelihood of descent.

The Leapfrog algorithm uses only gradient information in minimizing the objective function.

7.3.2 SQSD algorithm

The SQSD algorithm, developed by Snyman and Hay [14], makes successive spherical quadratic approximations to the objective function $f(\boldsymbol{x})$, as in Dynamic-Q (see Section 8.4). The second point $\boldsymbol{x}^1$ is calculated by stepping a distance δ in the steepest descent direction from a

given starting point $\boldsymbol{x}^0$:

$$\boldsymbol{x}^1 = \boldsymbol{x}^0 - \frac{\delta \nabla^T f(\boldsymbol{x}^0)}{||\nabla^T f(\boldsymbol{x}^0)||} \tag{7.5}$$

SQSD is used for unconstrained optimization and the solution of the quadratic subproblems can therefore explicitly be calculated as:

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k - \frac{1}{a^k} \nabla^T f(\boldsymbol{x}^k) \tag{7.6}$$

$$a^k = \frac{2\left[f(\boldsymbol{x}^{k-1}) - f(\boldsymbol{x}^k) - \nabla^T f(\boldsymbol{x}^k)(\boldsymbol{x}^{k-1} - \boldsymbol{x}^k)\right]}{||\boldsymbol{x}^{k-1} - \boldsymbol{x}^k||^2} \tag{7.7}$$

A sequence of points are generated using these equations (for $k = 1, 2....$). Move limits are added to ensure that no excessive change in position between iterations occur and the algorithm therefore converges in a stable manner. The maximum allowed step size between iterations is δ, i.e. if the combination of (7.6) and (7.7) results in:

$$||\boldsymbol{x}^{k+1} - \boldsymbol{x}^k|| > \delta \tag{7.8}$$

then $\boldsymbol{x}^{k+1}$ calculated with (7.6) is discarded and a new $\boldsymbol{x}^{k+1}$ is calculated by taking a step of size δ from $\boldsymbol{x}^k$ in the steepest descent direction:

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k - \frac{\delta \nabla^T f(\boldsymbol{x}^k)}{||\nabla^T f(\boldsymbol{x}^k)||} \tag{7.9}$$

7.4 Parallel implementation

In the multi-start competing algorithm approach, the minimizations performed by the algorithms are completely independent of each other and are therefore easily parallelized. As in Section 6.2, the pvm3 code [32] for FORTRAN is used enabling a cluster of computers to be used as a massive parallel processing virtual machine (MPPVM). Here again, the distributed computing model represents a master-slave configuration where the master program informs each slave task of the problem parameters and assigns optimization algorithms to the slaves. The master program awaits individual optimization results from each slave and parallel optimizations are terminated using (2.5). Appendix F.3 presents the master program code and Appendix F.4 the slave program code.

7.4.1 Assigning algorithms to slaves

Different strategies for assigning algorithms to the slaves are possible. For example, the algorithm assigned to a particular slave can be determined randomly with uniform probability, or according to a predetermined probability (based on, for instance, the performance of individual algorithms for a large set of test problems). The probabilities can also change during the analysis, according to the results obtained. In this initial study it is opted to use constant predetermined probabilities for the algorithms.

7.4.2 A measure of computational effort

Section 6.2.1 defined the *apparent visible cost* (N_{vc}) for the parallelized multi-start algorithms as the maximum function evaluations N_{fe} recorded for any slave. This definition for N_{vc} is not entirely satisfactory when gradient based and derivative free algorithms are used simultaneously. To facilitate the comparison between gradient based algorithms and derivative free algorithms, an algorithm's compound cost N_c is defined as $N_c = N_{fe} + nN_{ge}$, where N_{ge} represents the number of gradient evaluation performed, n the number of design variables and N_{fe} the number of function evaluations performed. Once again, it is assumed that the competing algorithm infrastructure will be used in problems for which the CPU requirements of evaluating the objective function is orders of magnitudes larger than the time required for message passing and algorithm internals.

Hence, the *apparent visible cost* (N_{vc}) of the parallelized competing algorithm infrastructure is defined as the greatest compound cost N_c recorded for any slave during the parallel optimization.

7.5 Results for parallel competing algorithms

Table 7.5 reflects the efficiency of the developed competing algorithm infrastructure. Here, GLS1, the Bayesian search implementation by Mockus, clustering, the Snyman-Fatti and the SQSD algorithms compete for a contribution to the stopping rule. The algorithms are assigned to the n_s slaves with an equal probability of 0.20. The computational effort associated with a single function evaluation and a gradient evaluation is increased artificially, as to prevent a bias towards algorithms which require little computational effort in evaluating algorithm internals. The lower limit on the number of random searches $\tilde{n}$ from the slaves is prescribed as 20 (compare Section 6.3).

While a specific algorithm in the infrastructure can yield superior results to those in Table 7.5 for a single given problem, the overall performance of the competing algorithm infrastructure for the complete test set is always superior. No failures were recorded when performing each test problem 7 times from different starting points.

Table 7.6 presents a breakdown of successful algorithms for a 128-node pvm. In the table, r_p indicates the proportional contribution to r, and $\tilde{n}_p$ the proportional contribution to $\tilde{n}$. High values for r_p indicate that an algorithm is successful in locating the global optimum f^*. Similarly, high values for $\tilde{n}_p$ indicate that an algorithm converged very quickly to non-optimal values. As expected, GLS1 converges quickly, resulting in high values of both r_p and $\tilde{n}_p$. This simple algorithm represents the largest contribution to r_p, with the exception of the Rastrigin problem, for which the Bayesian search strategy of Mockus makes the only notable contribution when n increases.

	GLS1					128-node pvm				
Prob.	N_{ge}	N_{fe}	N_c	$r/\tilde{n}$	$q(\tilde{n}, r)$	N_{ge}	N_{fe}	N_{vc}	$r/\tilde{n}$	$q(\tilde{n}, r)$
G1	3544	3544	10632	9 / 148	0.9990	0	337	337	9 / 119	0.9990
G2	3398	3398	37378	10 / 72	0.9990	5	1001	1051	10 / 40	0.9995
GP	764	764	2292	10 / 24	0.9996	4	97	105	10 / 32	0.9995
C6	330	330	990	10 / 16	0.9996	3	95	101	11 / 20	0.9998
SH	1724	1724	5172	10 / 65	0.9995	14	271	299	9 / 95	0.9990
RA ($n = 2$)	3010	3010	9030	9 / 154	0.9990	0	196	196	10 / 89	0.9995
RA ($n = 5$)	—	—	—	—	—	0	486	486	9 / 127	0.9990
RA ($n = 10$)	—	—	—	—	—	0	969	969	9 / 101	0.9990
BR	542	542	1626	9 / 9	0.9991	6	112	124	11 / 22	0.9998
H3	711	711	2844	10 / 17	0.9996	0	162	162	10 / 41	0.9995
H6	750	750	5250	9 / 14	0.9990	25	25	175	20 / 20	1.0000
S5	334	334	1670	10 / 23	0.9995	0	159	159	10 / 28	0.9995
S7	413	413	2065	10 / 27	0.9995	14	90	146	10 / 30	0.9995
S10	387	387	1935	10 / 24	0.9995	5	147	167	10 / 42	0.9995

Table 7.5: The results for the parallel competing algorithm infrastructure. The apparent visual cost N_{vc} for a 128 node parallel virtual machine may be compared with the compound cost N_c of the sequential GLS1 algorithm. '—' indicates that the GLS1 algorithm did not convergence to the global minimum f^*.

		Clustering	Mockus	GLS1	SF	SQSD
Prob.	$r/\tilde{n}$	$r_p/\tilde{n}_p$	$r_p/\tilde{n}_p$	$r_p/\tilde{n}_p$	$r_p/\tilde{n}_p$	$r_p/\tilde{n}_p$
G1	9 / 119	3 / 45	4 / 5	2 / 44	0 / 5	0 / 20
G2	10 / 40	6 / 12	0 / 0	4 / 28	0 / 0	0 / 0
GP	10 / 32	2 / 2	0 / 0	4 / 9	0 / 9	4 / 12
C6	11 / 20	2 / 2	0 / 0	6 / 9	0 / 0	3 / 9
SH	9 / 95	2 / 6	0 / 0	5 / 39	0 / 5	2 / 45
RA ($n = 2$)	10 / 89	0 / 2	6 / 6	2 / 40	0 / 2	2 / 39
RA ($n = 5$)	9 / 127	0 / 3	9 / 9	0 / 43	0 / 26	0 / 46
RA ($n = 10$)	9 / 101	0 / 1	9 / 9	0 / 45	0 / 0	0 / 46
BR	11 / 22	3 / 3	0 / 0	7 / 7	0 / 3	1 / 9
H3	10 / 41	4 / 4	1 / 12	0 / 8	0 / 0	5 / 17
H6	20 / 20	0 / 0	0 / 0	2 / 2	0 / 0	18 / 18
S5	10 / 28	0 / 1	1 / 4	7 / 17	0 / 0	2 / 6
S7	10 / 30	0 / 0	0 / 4	8 / 19	0 / 0	2 / 7
S10	10 / 42	1 / 2	0 / 9	6 / 20	0 / 0	3 / 11

Table 7.6: Breakdown of successful algorithms. r_p indicates the proportional contribution to r, and $\tilde{n}_p$ the proportional contribution to $\tilde{n}$.

7.6 Summary

A multi-start, multi-algorithm infrastructure is presented, in which different algorithms compete in parallel for a contribution towards the unified Bayesian global stopping criterion. The algorithm which is most suitable for a specific problem, outperforms the other algorithms and yields the largest contribution to the number of times the lowest minimum is found. This infrastructure is suitable for expensive unconstrained global programming problems.

Chapter 8

Constrained Global Optimization

8.1 Introduction

In addition to the unconstrained problem as formulated in Section 1.1, many practical problems include explicit constraint functions, which represent additional relationships among the variables. For instance, in structural optimization, the relationship between the variables should be such that the maximum allowable stress and the maximum allowable deflection are not exceeded.

The constrained optimization problem in general mathematical form is:

$$\text{minimize} \quad f(\boldsymbol{x}),\ \boldsymbol{x} = (x_1, x_2, ...x_n) \in R^n \tag{8.1}$$

subject to the constraints:

$$g_i(\boldsymbol{x}) \leq 0\ i = 1, 2, ...m \tag{8.2}$$

$$h_j(\boldsymbol{x}) = 0\ j = 1, 2, ...r \tag{8.3}$$

where f, g_i and h_j are scalar functions of $\boldsymbol{x}$.

This chapter presents three existing algorithms used for solving the constrained optimization problem, namely the LFOPC, ETOPC and Dynamic-Q algorithms. In addition, a constrained algorithm (GLS1C) using the search components of the GLS1 algorithm (see Chapter 3) is presented. The parallel infrastructure developed and presented in Chapters 6 and 7 is extended by the inclusion of these four constrained algorithms for solving expensive constrained optimization problems. The numerical results obtained using a set of simple constrained test functions are given here, for both serial and parallel implementations.

8.2 GLS1C

The performance of a constrained optimization algorithm, denoted GLS1C, using the search components of the GLS1 algorithm is investigated. The GLS1C constrained optimization algorithm consists of a global and local phase. The global phase is performed using the MBB algorithm on a penalty function formulation of the constrained problem. The solution of

the global phase is used as the starting point for the local phase which is performed with an augmented Lagrangian multiplier method. The BFGS algorithm is used for this local unconstrained optimization phase.

8.2.1 Penalty function method

The most simple and straight forward approach to handle constrained problems is with the penalty function method. The constrained problem is solved by applying an unconstrained optimization algorithm to a penalty function formulation $P(\boldsymbol{x})$ of the constrained problem:

$$\text{minimize } P(\boldsymbol{x}) \quad \text{where} \quad P(\boldsymbol{x}) = f(\boldsymbol{x}) + \sum_{i=1}^{m} b_i g_i^2(\boldsymbol{x}) + \sum_{j=1}^{r} w_j h_j^2(\boldsymbol{x}) \tag{8.4}$$

with:

$$b_i = \begin{cases} 0 & \text{if } g_i \leq 0 \\ s_i \gg 0 & \text{if } g_i > 0 \end{cases}$$

Usually the penalty parameters take on the same positive value, i.e. $s_i = w_j = p$. Very large values for p are used when high accuracy is needed, but the method unfortunately becomes unstable and inefficient for such high values of p.

8.2.2 Augmented Lagrangian multiplier methods

The Augmented Lagrangian multiplier method transforms the constrained problem to successive unconstrained problems involving the so-called Augmented Lagrangian. Combining the Lagrange function with the penalty function give the augmented Lagrange function L^k:

$$L^k(\boldsymbol{x}) = f(\boldsymbol{x}) + \sum_{i=1}^{m} \langle \frac{\lambda_i^k}{2p^k} + g_i(\boldsymbol{x}) \rangle^2 + p^k \sum_{j=1}^{r} h_j^2(\boldsymbol{x}) + \sum_{j=1}^{r} \lambda_j^k h_j(\boldsymbol{x}) \quad k = 0, 1, 2, 3... \tag{8.5}$$

where

$$\langle a \rangle = \text{maximum of } a \text{ and } 0 \tag{8.6}$$

The values for λ_i^k and λ_j^k are approximations to the Lagrange multipliers. The penalty parameter p^k is usually taken as a constant value p and low values for p result in a more stable method. The first multiplier approximations, λ_i^0 and λ_j^0, can arbitrarily be taken as 0. Any unconstrained optimization algorithm can be used to minimize the successive Lagrange functions with respect to $\boldsymbol{x}$. The solution $\boldsymbol{x}^{k-1}$ of Lagrange function L^{k-1} is used to calculate the next approximations to the Lagrange multipliers (λ_i^k and λ_j^k) in the following way:

$$\lambda_j^k = \lambda_j^{k-1} + 2p^{k-1} h_j(\boldsymbol{x}^{k-1}) \tag{8.7}$$

$$\lambda_i^k = < \lambda_i^{k-1} + 2p^{k-1} g_i(\boldsymbol{x}^{k-1}) > \tag{8.8}$$

where $\langle a \rangle$ = maximum of a and 0.

The successive optimizations are terminated if:

$$||\boldsymbol{x}^k - \boldsymbol{x}^{k-1}|| < \varepsilon_1 \quad \text{or} \tag{8.9}$$

$$||L^k(\boldsymbol{x}^k) - L^{k-1}(\boldsymbol{x}^{k-1})|| < \varepsilon_2 \tag{8.10}$$

8.3 LFOPC

The Leapfrog dynamic trajectory method developed by Snyman [36, 37] were modified to handle constraints using the penalty function formulation. This constrained version, called LFOPC, applies the penalty function formulation in 3 phases [38]:

Phase 0: For a given starting point $\boldsymbol{x}^0$ apply Leapfrog to the penalty function with $p := 100$, giving the solution $\boldsymbol{x}_0^*$ of phase 0.

Phase 1: Apply Leapfrog to the same penalty function but with $p =: 10000$, using $\boldsymbol{x}_0^*$ as the starting point. This gives $\boldsymbol{x}_1^*$ and the active inequality constraints at this point can be determined. The active inequality set $(g_a(\boldsymbol{x});\ a = 1, 2, ...na)$ corresponds to the set of inequality constraint functions for which $(g_i(\boldsymbol{x}_1^*) > 0;\ i = 1, 2, ...m)$.

Phase 2: The penalty function is redefined using only the active inequality set and the equality equations with $p = 10000$:

$$P_a(\boldsymbol{x}) = \sum_{a=1}^{na} pg_a^2(\boldsymbol{x}) + \sum_{j=1}^{r} ph_j^2(\boldsymbol{x}) \tag{8.11}$$

Leapfrog is used for minimizing $P_a(\boldsymbol{x})$ with $\boldsymbol{x}_1^*$ as starting point, giving the solution $\boldsymbol{x}^*$ to the constrained problem.

8.4 Dynamic-Q

The Dynamic-Q method [40] consists of applying the LFOPC algorithm to successive approximate quadratic subproblems $(P[k], k = 0, 1, 2, 3...)$ derived from the actual problem functions. The Dynamic-Q method gives results of equal accuracy within comparable number of function evaluations when compared to the performance of the sequential quadratic programming (SQP) methods [41].

The Dynamic-Q method is primarily intended for constrained optimization problems where function evaluations are expensive. The actual problem functions, $f(\boldsymbol{x})$, $g_i(\boldsymbol{x})$ and $h_j(\boldsymbol{x})$, are approximated by spherically quadratic functions, $\tilde{f}(\boldsymbol{x})$, $\tilde{g}_i(\boldsymbol{x})$ and $\tilde{h}_j(\boldsymbol{x})$ at design point $\boldsymbol{x}^k$ as follows:

$$\tilde{f}(\boldsymbol{x}) = f(\boldsymbol{x}^k) + \boldsymbol{\nabla}^T f(\boldsymbol{x}^k)(\boldsymbol{x} - \boldsymbol{x}^k) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^k)^T \boldsymbol{A}^k (\boldsymbol{x} - \boldsymbol{x}^k) \tag{8.12}$$

$$\tilde{g}_i(\boldsymbol{x}) = g_i(\boldsymbol{x}^k) + \boldsymbol{\nabla}^T g_i(\boldsymbol{x}^k)(\boldsymbol{x} - \boldsymbol{x}^k) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^k)^T \boldsymbol{B}_i^k (\boldsymbol{x} - \boldsymbol{x}^k) \quad i = 1, 2, ...m \tag{8.13}$$

$$\tilde{h}_j(\boldsymbol{x}) = h_j(\boldsymbol{x}^k) + \boldsymbol{\nabla}^T h_j(\boldsymbol{x}^k)(\boldsymbol{x} - \boldsymbol{x}^k) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^k)^T \boldsymbol{C}_j^k (\boldsymbol{x} - \boldsymbol{x}^k) \quad j = 1, 2, ...r \tag{8.14}$$

with the diagonal Hessian matrices $\boldsymbol{A}^k$, $\boldsymbol{B}_j^k$ and $\boldsymbol{C}_j^k$:

$$\boldsymbol{A}^k = a^k \boldsymbol{I} \tag{8.15}$$

$$\boldsymbol{B}_i^k = b_i^k \boldsymbol{I} \tag{8.16}$$

$$\boldsymbol{C}_j^k = c_j^k \boldsymbol{I} \tag{8.17}$$

Using these quadratic equations, subproblem $P[k]$ are defined as:

$$\text{minimize} \quad \tilde{f}(\boldsymbol{x}), \ \boldsymbol{x} = (x_1, x_2, ...x_n) \in R^n \tag{8.18}$$

subject to:

$$\tilde{g}_i(\boldsymbol{x}) \leq 0 \ i = 1, 2, ...m \tag{8.19}$$

$$\tilde{g}_{m+1}(\boldsymbol{x}) = ||\boldsymbol{x} - \boldsymbol{x}^k||^2 - \delta^2 \leq 0 \tag{8.20}$$

$$\tilde{h}_j(\boldsymbol{x}) = 0 \ j = 1, 2, ...r \tag{8.21}$$

An additional constraint $\tilde{g}_{m+1}(\boldsymbol{x})$ is added which ensures that the maximum step between iterations are δ. The curvatures a^0, b_i^0 and c_j^0 for the first subproblem $P[0]$ is taken as zero, forming linear functions. Solving the first subproblem $P[0]$ for any starting point $\boldsymbol{x}^0$ gives $\boldsymbol{x}^1$. Thereafter the curvatures for subproblems $P[k]; k = 1, 2, 3, 4...$ are defined as follows:

$$a^k = \frac{2\left[f(\boldsymbol{x}^{k-1}) - f(\boldsymbol{x}^k) - \boldsymbol{\nabla}^T f(\boldsymbol{x}^k)(\boldsymbol{x}^{k-1} - \boldsymbol{x}^k)\right]}{||\boldsymbol{x}^{k-1} - \boldsymbol{x}^k||^2} \tag{8.22}$$

$$b_i^k = \frac{2\left[g_i(\boldsymbol{x}^{k-1}) - g_i(\boldsymbol{x}^k) - \boldsymbol{\nabla}^T g_i(\boldsymbol{x}^k)(\boldsymbol{x}^{k-1} - \boldsymbol{x}^k)\right]}{||\boldsymbol{x}^{k-1} - \boldsymbol{x}^k||^2} \tag{8.23}$$

$$c_j^k = \frac{2\left[h_j(\boldsymbol{x}^{k-1}) - h_j(\boldsymbol{x}^k) - \boldsymbol{\nabla}^T h_j(\boldsymbol{x}^k)(\boldsymbol{x}^{k-1} - \boldsymbol{x}^k)\right]}{||\boldsymbol{x}^{k-1} - \boldsymbol{x}^k||^2} \tag{8.24}$$

A sequence of points are found by using the solution of subproblem $P[k]$ as the next point $\boldsymbol{x}^{k+1}$ at which the following subproblem $P[k+1]$ is constructed.

8.5 ETOPC

The penalty function problems associated with large values for the penalty parameter p, can be solved by applying Sequential Unconstrained Minimization Techniques (SUMT) [42]. SUMT consists of performing a succession of penalty function $P^k(\boldsymbol{x})$ minimizations, with the penalty parameter p^k increasing with each minimization. Snyman used SUMT with the ETOP [43] unconstrained algorithm in developing the ETOPC constrained algorithm. More formally, the ETOPC algorithm consists of the following steps:

1. Given a starting point $\boldsymbol{x}^0$, set $k := 0$ and choose the first penalty parameter p^1.

2. Set $k := k + 1$ and minimize $P^k(\boldsymbol{x})$ with associated penalty parameter p^k and starting point $\boldsymbol{x}^{k-1}$, using ETOP to give $\boldsymbol{x}^k$.

3. Test convergence criteria. If:

$$||\boldsymbol{x}^k - \boldsymbol{x}^{k-1}|| < \varepsilon_1 \quad \text{or} \tag{8.25}$$

$$||P^k(\boldsymbol{x}^k) - P^{k-1}(\boldsymbol{x}^{k-1})|| < \varepsilon_2 \tag{8.26}$$

then STOP, else continue.

4. Set $p^{k+1} := cp^k$ with $c > 1$ and go to 2.

p^1 and c are typically taken as 1 and 10 respectively.

8.6 Constrained algorithms in multi-start procedure

In Chapter 3 efficient multi-start algorithms are presented to solve unconstrained global programming problems. The constrained global programming problem is addressed in this section using the same multi-start strategy. It is argued that the constrained or unconstrained global minimum is associated with a higher probability of convergence than any other constrained or unconstrained local minimum. The unified Bayesian stopping rule is therefore also applicable to constrained algorithms.

8.6.1 Numerical results

The above mentioned constrained algorithms are combined with the unified Bayesian stopping rule in a multi-start procedure and tested on a set of simple constrained test functions. Appendix B presents the 7 problems explicitly.

The stopping rule parameters used are $a = 1$, $b = 5$ and $q^* = 0.999$. Table 8.1 shows the average number of function evaluations N_{fe}, average number of gradient evaluations N_{ge} and the number of failures F to converge to the constrained optima for 10 independent runs of each problem using each method. The table reveals that the Dynamic-Q method outperformes the other algorithms. This is expected as the test functions used are mainly of quadratic form. For the Dynamic-Q algorithm, the value of δ was chosen as 100 for all the problems, except for C6 where $\delta = 1$ was used.

The suitability of the unified Bayesian stopping rule is also demonstrated in that no failures are recorded for all the problems and algorithms, except for GLS1C when applied to problem C3.

	GLS1C			LFOPC			ETOPC			Dynamic-Q		
Prob.	N_{fe}	N_{ge}	F	N_{fe}	N_{ge}	F	N_{fe}	N_{ge}	F	N_{fe}	N_{ge}	F
C1	433	433	0	37	1490	0	144	906	0	90	90	0
C2	2180	2180	0	36	3757	0	157	10389	0	27	27	0
C3	8251	8251	2	54	6012	0	164	7927	0	153	153	0
C4	810	810	0	36	1520	0	144	11706	0	64	64	0
C5	150	150	0	37	2080	0	144	5800	0	64	64	0
C6	3772	3772	0	54	12488	0	180	6667	0	483	483	0
C7	21661	21661	0	64	44210	0	294	64260	0	328	328	0

Table 8.1: Results for constrained algorithms combined with the unified Bayesian stopping criterion in multi-start procedures.

8.7 Multiple parallel constrained searches

In Chapter 6 it is shown that multiple independent searches in a multi-start procedure performed simultaneously on different computers, effectively reduces the cost of solving expensive global programming problems. The same approach is adopted for expensive constrained global programming problems. For demonstration purposes, the efficient Dynamic-Q algorithm is used in the multi-start parallel infrastructure as explained in Chapter 6. Table 8.2 shows the results when solving the constrained test set using a MPPVM consisting of 32 computers. The table shows that the computational effort of the parallel multi-start Dynamic-Q algorithm is much less than the computational effort of the sequential multi-start Dynamic-Q algorithm (shown in Table 8.1). For the parallel multi-start Dynamic-Q algorithm the results from a minimum of 20 sampling points $\tilde{n}$ are required before stopping. This increases the probability of finding the global minimum from 0.999 for the sequential algorithm to the probability q^* shown in the last column of Table 8.2, without increasing

Prob.	N_{vc}	$r/\tilde{n}$	$q(\tilde{n}, r)$
C1	17	20 / 20	1.0000
C2	3	20 / 20	1.0000
C3	16	10 / 24	0.9995
C4	10	20 / 20	1.0000
C5	8	11 / 21	0.9998
C6	70	18 / 20	1.0000
C7	56	20 / 20	1.0000

Table 8.2: Apparent visual cost N_{vc} and the probability $q(\tilde{n}, r)$ that $\tilde{f}$ is equal to f^* for the parallel multi-start Dynamic-Q algorithm using a MPPVM consisting of 32 machines. The apparent visual cost N_{vc} for a 32-node parallel virtual machine may be compared with the number of function evaluations N_{fe} of the sequential Dynamic-Q algorithm.

the computational cost.

8.8 Parallel competing constrained algorithms

The unconstrained global programming problem is addressed in Chapter 7 by simultaneously applying different unconstrained algorithms using a MPPVM. The same approach is adopted here for constrained global programming problems using the four constrained optimization algorithms, GLS1C, LFOPC, ETOPC and Dynamic-Q. These algorithms compete for a contribution to the unified Bayesian stopping rule as described in Chapter 7. In this case, the algorithms are assigned to the n_s slaves with an equal probability of 0.25.

The *apparent visible cost* (N_{vc}) as defined in Section 7.4.2 is used to express the performance of the competing constrained algorithms method. Table 8.3 shows the function evaluations N_{fe} and the gradient evaluations N_{ge} that make up the greatest compound cost N_c recorded for a specific slave, when solving the constrained test set with a 32-node MPPVM. Table

Prob.	N_{ge}	N_{fe}	N_{vc}	$r/\tilde{n}$	$q(\tilde{n}, r)$
C1	51	41	143	20 / 20	1.0000
C2	175	151	851	17 / 20	1.0000
C3	182	37	583	10 / 26	0.9995
C4	95	66	256	20 / 20	1.0000
C5	22	22	132	15 / 20	1.0000
C6	308	137	2293	13/ 20	1.0000
C7	574	426	4444	15/ 20	1.0000

Table 8.3: Apparent visual cost N_{vc} consisting of the gradient evaluations N_{ge} and the function evaluations N_{fe} for the parallel competing algorithm method using a 32-node MPPVM.

		GLS1C	LFOPC	ETOPC	Dynamic-Q
Prob.	$r/\tilde{n}$	$r_p/\tilde{n}_p$	$r_p/\tilde{n}_p$	$r_p/\tilde{n}_p$	$r_p/\tilde{n}_p$
C1	20/ 20	4 / 4	0 / 0	1 / 1	15 / 15
C2	17/ 20	4 / 7	0 / 0	0 / 0	13 / 13
C3	10/ 26	0 / 9	2 / 2	0 / 0	8 / 15
C4	20/ 20	4 / 4	1 / 1	0 / 0	15 / 15
C5	15/ 20	7 / 7	0 / 0	0 / 0	8 / 13
C6	13/ 20	2 / 7	0 / 0	1 / 2	10 / 11
C7	15/ 20	1 / 6	0 / 0	0 / 0	14 / 14

Table 8.4: The contributions of the algorithms towards the unified Bayesian stopping rule. r_p indicates the proportional contribution to r, and $\tilde{n}_p$ the proportional contribution to $\tilde{n}$.

8.4 shows the contributions of the algorithms towards the stopping rule.

From the results presented in Table 8.1 it is expected that the Dynamic-Q algorithm will have the greatest contribution, and the results of Table 8.4 confirm this. The Dynamic-Q algorithm outperformed the other algorithms and the high values of r_p indicate that the algorithm was successful in locating the global optimum f^* a number of times.

8.9 Summary

The global constrained problem is addressed in the following ways:

- Constrained algorithms are combined with the unified Bayesian stopping rule in effective multi-start algorithms.
- The independent searches involved in the multi-start procedure are performed simultaneously on different computers, which effectively reduces the computational cost.
- Using different constrained algorithms simultaneously in a multi-start procedure, results in a highly robust and efficient method for solving expensive constrained global programming problems.

Chapter 9

Slope Stability Analysis

9.1 Introduction

Calculating the safety factor of slopes is important in a number of engineering applications. These include natural slopes, earthworks construction, embankments, earth dams, etc. In recent years finite element methods have been developed for slope stability analyses [3, 4], but limiting equilibrium methods are still widely used.

Limiting equilibrium methods require an assumption about the geometry of the failure plane. For homogeneous soils the critical failure plane can be accurately modelled using mathematical functions such as circular arcs or log spiral functions, but for layered soils the geometry of the failure plane can become irregular. Methods which assume a failure plane of a regular shape are widely used for slope stability analyses. A computer can be used to randomly generate hundreds of different failure planes and identify the one with the smallest factor of safety as the critical plane. However, this method is not suitable for non-homogeneous soil profiles. With a complex soil profile the shape and position of the failure plane can not be modelled by a simple mathematical function and an alternative method for finding the critical failure plane is needed.

Limiting equilibrium methods such as Janbu's method [44, 45], Morgenstern Price's method [46] and Spencer's method [47, 48] are suitable to determine the factor of safety for an arbitrary failure surface geometry. The problem of finding the critical failure plane when using these methods may be addressed by randomly generating possible failure planes, but with some restrictions to ensure the kinematically admissibility of the surfaces [49]. Nguyen [50] developed a method where the factor of safety is formulated as a multivariate function $F(\boldsymbol{x})$ with the independent variables $\boldsymbol{x}$ describing the geometry of the failure plane, which can be circular or non-circular. He employed the simplex method as optimization technique. Celestina and Duncan [51] used the same approach for non-circular failure planes, but used the alternating-variable optimization technique. Li [52] proposed a more efficient one-dimensional optimization technique to replace the quadratic interpolation method which Celestina and Duncan [51] used in the alternating-variable technique. Baker [53] made use of Spencer's method and defined the failure plane by a number of nodal points connected by straight lines. The vertical coordinates of the nodal points are the variables in Baker's

method and the dynamic programming technique is employed as the optimization method. Recently, workers have addressed the problem by using a genetic algorithm that determines the critical slip surface for a multiple wedge stability analysis [23].

Nguyen's [50] method for non-circular failure planes and Baker's method [53] are fairly general, but lack the property of variable horizontal coordinates of the nodal points in the analyses. Celestina and Duncan's [51] method is formulated such that the horizontal coordinates of the nodal points may vary, but some of the coordinates are given a prescribed direction of movement. This inhibits the failure plane to be entirely general. It is also uncertain if the algorithms used for the different methods are able to locate the global minimum of the factor of safety formulations. For some of these methods the starting values for the variables are chosen such that the initial failure plane closely resembles the critical plane, thereby reducing the effort of finding the global minimum.

This chapter presents a global optimization algorithm combined with Janbu's simplified method and Spencer's method as an alternative method for finding the critical failure plane of any shape. The failure plane is defined by nodal points connected by straight lines. The points are equidistant from each other with the first and last points at the ground surface. These two points may move horizontal and consequently all the nodal points have the possibility of moving horizontal together with the freedom of vertical movement. This results in a general formulation of the failure plane position.

The next section briefly revises Janbu's and Spencer's method and presents the relevant equations to calculate the factor of safety using these two methods. The unconstrained problem is then formulated using these equations and the optimization algorithm used for solving the problem is discussed. Four examples are presented to demonstrate the performance of the new slope stability optimization procedure.

9.2 Janbu's simplified method

Janbu's simplified method [44, 45] forms part of the limiting equilibrium techniques based on the method of slices. This method was developed to calculate the factor of safety for slip surfaces of any shape. Firstly a failure plane is assumed and the slipping mass is divided into vertical slices. The following static equilibrium conditions of each slice is considered:

- Sum of the vertical forces $\sum F_y = 0$
- Sum of the forces parallel to failure plane $\sum F_{||} = 0$

For the soil mass as a whole the equations used are:

- Sum of the vertical forces $\sum F_y = 0$
- Sum of the horizontal forces $\sum F_x = 0$

The above equations indicate that Janbu's simplified method may violate moment equilibrium of the soil mass as a whole.

The factor of safety is defined as:

$$F = \frac{\text{soil shear strength}}{\text{mobilized shear stress}} \tag{9.1}$$

Combining these equations gives Janbu's simplified factor of safety as:

$$F = \frac{\sum_{i=1}^{n}(c_i b_i + (W_i - u_i b_i)\tan\phi_i)/(m_\alpha \cos\alpha_i)}{\sum_{i=1}^{n} W_i \tan\alpha_i} \tag{9.2}$$

where:

$$m_\alpha = \left[1 + \frac{\tan\alpha_i \tan\phi_i}{F}\right] \tag{9.3}$$

and
α_i = angle of failure plane for slice i
W_i = weight of slice i
ϕ_i = angle of internal friction slice i
c_i = cohesion of slice i
u_i = water pressure of slice i
n = number of vertical slices.

Note that F is present on both sides of (9.2). The secant iterative method which finds the roots of a function $g(F)$ with one variable is used to solve F. $g(F)$ is defined as:

$$g(F) = \left[\frac{\sum_{i=1}^{n}[(c_i b_i + (W_i - u_i b_i)\tan\phi_i)/(m_\alpha \cos\alpha_i)]}{\sum_{i=1}^{n} W_i \tan\alpha_i}\right] - F \tag{9.4}$$

and the secant formula is given by:

$$F_{k+1} = F_k + \frac{g(F_k)\cdot(F_k - F_{k-1})}{g(F_k) - g(F_{k-1})} \quad \text{for} \quad k = 1, 2, 3, ... \tag{9.5}$$

The above equations tend to converge rapidly.

9.3 Spencer's method

Spencer [47, 48] developed a slope stability analysis technique based on the method of slices, which satisfies all equilibrium equations. Spencer's method is suitable for a failure surface of arbitrary shape and considers the following equilibrium conditions for each slice:

- Sum of the forces perpendicular to failure plane $\sum F_\perp = 0$
- Sum of the forces parallel to failure plane $\sum F_{||} = 0$
- Sum of the moments about the middle of the slice's base $\sum M = 0$

Combing these equations with (9.1) results in the following two equations:

$$H_1(F,\theta) = \sum_{i=1}^{n} \left[\frac{c_i b_i \sec\alpha_i / F + \tan\phi_i (W_i \cos\alpha_i - u_i b_i \sec\alpha_i)/F - W_i \sin\alpha_i}{\cos(\alpha_i - \theta)\,[1 + \tan\phi_i \tan(\alpha_i - \theta)/F]} \right] = 0 \qquad (9.6)$$

$$H_2(F,\theta) = \sum_{i=2}^{n} \left[\frac{1}{2} Z_{i-1} (\sin\theta (b_i + b_{i-1}) - \cos\theta (b_i \tan\alpha_i + b_{i-1} \tan\alpha_{i-1}) \right] = 0 \qquad (9.7)$$

where:

$$Z_{i-1} = \frac{c_i b_i \sec\alpha_i / F + \tan\phi_i (W_i \cos\alpha_i - u_i b_i \sec\alpha_i)/F - W_i \sin\alpha_i}{\cos(\alpha_i - \theta)\,[1 + \tan\phi_i \tan(\alpha_i - \theta)/F]} + Z_{i-2} \qquad (9.8)$$

and
$Z_0 = 0$
θ = direction of interslice force.

(9.6) considers force equilibrium and (9.7) moment equilibrium, with the two unknowns F and θ. The factor of safety of the specified failure plane is taken as F, with the corresponding θ, which satisfies these two equations. The problem of solving the set of two non-linear equations is transformed to a minimization problem by defining an auxiliary function $G(F,\theta)$:

$$G(F,\theta) = [H_1(F,\theta)]^2 + [H_2(F,\theta)]^2 \qquad (9.9)$$

Finding the values for F and θ that results in $G(F,\theta) = 0$ corresponds to the solution of the two equations. The BFGS algorithm [10, 11, 12] is used for this minimization phase.

9.4 Mathematical representation of failing mass

Janbu's and Spencer's factor of safety equations require information regarding each slice. These are the weight, pore water pressure at the failure plane, angle of internal friction, cohesion and the failure plane angle. These values are dependent on the geometry of the failure plane, the width and the number of slices. The analysis therefore requires a unique set of variables $\boldsymbol{x}$ representing a failure plane of any shape.

For the method described herein, the first two variables x_1 and x_2, defines the horizontal positions where the failure plane intersects the ground surface (see Figure 9.1). x_1 is the horizontal distance between a reference point and the initiation point of the failure plane, while the horizontal distance between the initiation and termination points of the failure plane is denoted as x_2. The method of slices requires the failing soil mass to be divided into n vertical slices and the bottom of each slice forms the failure plane. The vertical distances of the slice interfaces, measured from the ground level to the failure plane, can be used to define a unique failure plane in terms of the variables $(x_3, x_4, x_5, ...x_{n+1})$. The failure plane between the slice interfaces is assumed to be linear. In this analysis the width of the failing mass x_2 is divided into n slices of equal width and the width of each slice may be calculated as:

$$b_i = \frac{x_2}{n} \qquad (9.10)$$

The formulation of the unconstrained problem is completed with $n+1$ independent variables $\boldsymbol{x}$ describing the failure plane and the factor of safety $F(\boldsymbol{x})$ as the objective function.

9.5 Slope stability optimization procedure

The procedure for finding the critical slip surface consists of two phases, a global optimization phase and a local refinement phase. In the global optimization phase multiple independent searches are performed with a minimum number of slices to determine an initial approximation to the critical slip surface. The local refinement phase uses the approximate solution of the global optimization phase to refine the slip surface geometry. This strategy is used to reduce the overall number of function evaluations required.

More formally the slope stability optimization procedure is as follows:

1. **Initialization:** Set the counter $j := 1$, prescribe the desired confidence level q^*, t_{max}, n_1, k_{max}, l_{max} and x_{beg}. Here, x_{beg} denotes the maximum random starting value for $x_3, x_4, x_5, ...x_{n_k+1}$, t_{max} the maximum number of global phase iterations, n_1 the starting number of slices, k_{max} the maximum number of adaptive slicing loops in the global phase and l_{max} the maximum number of adaptive slicing loops in the local phase.

2. **Global Optimization phase:**

   (a) **Sampling steps:** Set the counter $k := 1$ and start with n_k slices and randomly generate $\boldsymbol{x}_k^j \in D$. That is choose x_1 and x_2 randomly within the slope geometry and generate random values for $x_3, x_4, x_5, ...x_{n_k+1}$ between 0 and depth x_{beg}.

   (b) **Minimization steps:** Starting at $\boldsymbol{x}_k^j$, attempt to minimize F in a global sense by any optimization algorithm, viz. find and record some low function value $\tilde{F}_k^j \leftrightarrow \tilde{\boldsymbol{x}}_k^j$.

   (c) **Termination check:** If $k = k_{max}$ or $\tilde{F}_k^j \geq 10$ go to 3, else continue.

   (d) **Double number of slices:** Set $k := k + 1$, double the number of slices ($n_k := 2n_{k-1}$) and determine the new starting vector $\boldsymbol{x}_k^j$ from $\tilde{\boldsymbol{x}}_{k-1}^j$ (perform adaptive slicing as explained in Section 9.5.1). Go to 2 (b).

3. **Global Termination:** Assess the global convergence after $\tilde{n}$ searches was completed (yielding $\tilde{\boldsymbol{x}}_k^j$ and $\tilde{F}_k^j, j = 1, 2, ..., \tilde{n}$) using (2.4). If (2.5) is satisfied or $j = t_{max}$, go to 4, else $j := j + 1$ and go to 2.

4. **Local refinement phase:**

   (a) **Initialization:** Set the counter $l := 2$ and determine the starting vector $\bar{\boldsymbol{x}}_l$ for the local refinement phase from $\tilde{\boldsymbol{x}}_k^j$ (see Section 9.5.1) which corresponds to the lowest recorded $\tilde{F}_k^j$ for $j = 1, 2, ..., \tilde{n}$. Set $\hat{F}_1 = \tilde{F}_k^j$ and the number of slices are $n_l := 2n_k^j$.

   (b) **Minimization steps:** Starting at $\bar{\boldsymbol{x}}_l$, attempt to minimize F in a local sense by any optimization algorithm, viz. find and record some low function value $\hat{F}_l \leftrightarrow \hat{\boldsymbol{x}}_l$.

   (c) **Termination check:** If $l = l_{max}$ or $\hat{F}_l > \hat{F}_{l-1}$ go to 5, else continue.

   (d) **Double number of slices:** Set $l := l + 1$, double the number of slices ($n_l := 2n_{l-1}$) and determine the new starting vector $\bar{\boldsymbol{x}}_l$ from $\hat{\boldsymbol{x}}_{l-1}$ (perform adaptive slicing as explained in Section 9.5.1). Go to 4 (b).

5. **Slope Stability Termination:** Take the lowest recorded $\hat{F}_l$ for $l = 1, 2, 3...$ as factor of safety. STOP.

Typical search routines require the approximate position of the initiation and the termination points of the failure plane to be specified. This requires *a priori* knowledge of the behaviour of soil slopes. Step 2 (a) requires no *a priori* information regarding the initiation and the termination points and results in the procedure being more general. The values for $x_3, x_4, x_5, ...x_{n_k+1}$ are not constrained within bounds during the analysis. Also note that the unified Bayesian stopping criterion is utilized to terminate the global optimization phase mentioned in step 3.

9.5.1 Adaptive slicing

The minimum found from an optimization iteration with n slices can be used to determine the starting point for the next optimization iteration with double the number of slices ($2n$ slices). This is done by introducing another slice interface in the centre of each of the n slices and by linear interpolation of the failure surface the values for the new intermediate variables can be obtained. The starting values for the variables $(x_1, x_2, x_3, ...x_{2n+1})$ therefore presents the exact same failure plane as was found by the n-slice solution. The motivation for this is reduced computational effort.

9.6 Optimization algorithms

A number of algorithms were tested as part of the described slope stability optimization procedure. These include GLS1 (Chapter 3), Leapfrog [38], ETOP [43], a GA (Chapter 4) and the PSOA presented in Chapter 5. The Leapfrog algorithm proved to be the most efficient algorithm for this problem and the results of the Leapfrog algorithm are described below in more detail. Results for Janbu's and Spencer's method combined with the Leapfrog algorithm are presented, and are denoted Leapfrog-Janbu and Leapfrog-Spencer.

9.6.1 Leapfrog algorithm

As mentioned in Section 7.3.1, the Leapfrog [38] algorithm uses only gradient information in minimizing the objective function. The defined objective functions have no explicitly defined gradient functions and the gradient vectors are calculated with first order difference formulas.

9.7 Examples

Three non-homogeneous examples are taken from Goh [23] and Fredlund and Krahn [54] to illustrate the performance of the algorithms with Janbu's and Spencer's method in the slope

Soil No.	Cohesion (kPa)	Friction angle (°)	Unit Weight (kN/m^3)
1	0.0	38.0	19.5
2	5.3	23.0	19.5
3	7.2	20.0	19.5

Table 9.1: Soil parameters for Example 1.

Soil No.	Cohesion (kPa)	Friction angle (°)	Unit Weight (kN/m^3)
1	28.73	20.0	18.84
2	0.0	10.0	18.84

Table 9.2: Soil parameters for Example 2.

stability procedure. A forth example is used to demonstrate the effectiveness of the slope stability procedure for multiple slope sections. The safety factors of the examples reported by Goh [23] are compared with the results obtained using the new analysis technique.

The soil slope for Example 1 consists of three layers, as shown in Figure D.1. The soil parameters are given in Table 9.1. For Example 2 the soil profile contains a 0.5m weak layer as depicted in Figure D.2. Table 9.2 tabulates the soil parameters of the different layers. The soil profile of Example 3 (Figure D.3) contains an inclined weak layer of thickness 1m, with properties given in Table 9.3. Example 4 represents a multiple slope geometry and the dimensions are those typically used for gold tailing dams [55] (see Figure D.4). The problem contains a phreatic surface with the unit weight of water taken as 9.81 kN/m^3.

The parameter values used in the slope stability optimization procedure are q^*=0.999, t_{max} = 100, k_{max} = 3, l_{max} = 3 and x_{beg} = 15m. n_1 = 4 for the Leapfrog-Janbu analysis, while n_1 = 3 for the Leapfrog-Spencer analysis.

9.8 Discussion of numerical results

The safety factors of the critical failure planes found by the Leapfrog algorithm using Janbu's and Spencer's method are presented in Table 9.5. In addition, the coordinates which define the critical slip surfaces are given in Appendix E, while Figures D.5 to D.12 depict these critical failure planes. Table 9.6 presents the safety factors using the methods reported by Goh [23]. The function evaluations performed using Leapfrog in the different analyses are given in Table 9.7.

It is important to note that the governing equations for the methods presented in the tables are different. Janbu's method ignores interslice shear forces and violates moment equilibrium for the mass as a whole. In contrast, Spencer's method incorporates the interslice shear force by assuming a constant force angle and satisfies all equilibrium conditions. This results in Janbu's method generating factors of safety lower than those calculated using Spencer's

Soil No.	Cohesion (kPa)	Friction angle (°)	Unit Weight (kN/m^3)
1	10.0	25.0	20.0
2	0.0	10.0	20.0

Table 9.3: Soil parameters for Example 3.

Soil No.	Cohesion (kPa)	Friction angle (°)	Unit Weight (kN/m^3)
1	5.0	35.0	17.0
2	5.0	35.0	19.0

Table 9.4: Soil parameters for Example 4.

method (see for example Fredlund and Krahn [54]). For this reason Janbu [56, 57] suggested a correction factor. As expected, using Janbu's method the Leapfrog algorithm calculated lower factors of safety compared with Spencer's method, but the critical failure planes calculated using the two methods are only slightly different (see Figures D.5 to D.12). This generates confidence that the global optima were found.

Tables 9.5 and 9.6 allows the comparison of the factors of safety presented by Goh [23] to those calculated using this new two phase technique. The methods reported by Goh [23] (shown in Table 9.6), make different assumptions and use different equations. The global minimum of each method will not necessarily be equal even though the same slope problem is analysed and even when the same optimization algorithm is used. The Leapfrog-Janbu values are lower for all the examples. This was due to the algorithm being able to find more critical failure planes, but as explained also to an extent that Janbu's method tends to be conservative. Using Spencer's method the Leapfrog algorithm found a lower value for Example 1 compared with the result for Spencer's method presented by Goh [23].

The algorithm satisfied the stopping rule (2.5) before the maximum of 100 iterations were reached only for Example 1. The number of function evaluations for Examples 2, 3 and 4 are therefore less to attain the prescribed confidence level q^*=0.999.

Table 9.7 shows the number of function evaluations for the different methods. Janbu's method is more costly to solve in terms of function evaluations, but a single function evaluation with Spencer's method far exceed the time needed to perform a Janbu evaluation. The Leapfrog-Janbu analysis for Example 1 performed 21136 Janbu function evaluations and took 14 seconds with a Pentium III 800 MHz processor. The average time for a single function evaluation was therefore 0.0007 seconds. Performing the Leapfrog-Spencer analysis on the same problem took 453 seconds and 30877 function evaluations to complete. A single function evaluation was therefore performed in 0.015 seconds. The time of a Spencer function evaluation is therefore almost 22 times longer than that for a Janbu function evaluation. So even though the number of Janbu function evaluations exceeds Spencer's function evaluations, the time for a Janbu's analysis is significantly less.

Method	Example 1	Example 2	Example 3	Example 4
Leapfrog-Janbu	1.247	1.195	0.879	1.476
Leapfrog-Spencer	1.359	1.305	1.060	1.549

Table 9.5: Factor of safety using the Leapfrog algorithm in slope stability optimization procedure.

Method	Example 1	Example 2	Example 3
GAWEDGEM(6)	1.387	1.288	1.021
GAWEDGE(6)	1.393	1.286	1.003
Slope	—	1.364-1.378	—
Spencer	1.39	1.24	—
Chen and Shoa	1.39	1.242	—
Donald and Giam	1.39	1.27	—

Table 9.6: Factor of safety calculated with methods reported by Goh.

9.9 Recommendations

For the results presented in the tables no restrictions were placed on the starting values for the initiation point x_1 and the termination point $(x_1 + x_2)$ of the failure plane. For single slope problems (Examples 1 to 3), more reasonably chosen starting values (Step 2 (a)) for x_1 and x_2 will improve the procedure's efficiency. The first initiation point can randomly be selected within an interval of appropriately chosen size containing the slope's toe. The first termination point can randomly be selected within a similarly chosen interval containing the crest of the slope. This pre-knowledge of the possible position of the critical failure plane can therefore be used to reduce the effort to solve the problem. For multiple slope geometries, as in Example 4, the position of the initiation and termination points are difficult to predict and ideally no restrictions should be placed on the starting values for x_1 and x_2. The new method described above does not place any restrictions on the values of x_1 and x_2 and are therefore most suitable to find the global minimum factor of safety for multiple slope geometries.

Spencer's function evaluation time far exceeds Janbu's function evaluation time. Therefore, it seems practical to perform the global optimization phase (Step 2) with Janbu's method and the local refinement phase (Step 4) with Spencer's method. The analysis time and number of function evaluations with Spencer's method will therefore be significantly less.

9.10 Summary

This chapter describes a global optimization procedure for calculating the critical failure plane in slope stability analyses using Janbu's simplified method or Spencer's method of

Method	Example 1	Example 2	Example 3	Example 4
Leapfrog-Janbu	21136	98727	125794	31784
Leapfrog-Spencer	30877	24269	38025	27018

Table 9.7: Number of function evaluations using the Leapfrog algorithm for the two methods.

slices. The procedure starts with a global phase where a number of independent runs with a few slices are performed to determine an initial approximation to the critical failure plane. The procedure then refines this approximate surface with increased number of slices. Any optimization algorithm can be employed in the procedure but in this study the Leapfrog algorithm is used.

The safety factors obtained with this procedure using Janbu's method and Spencer's method are slightly lower than the reported values for the three examples considered. As expected, Janbu's method gives more conservative results when compared with Spencer's method. The time needed for a single Spencer analysis is significantly longer than a Janbu analysis and the number of function evaluations becomes important only when Spencer's method is used.

A slope geometry consisting of multiple inclined sections demonstrates the robustness of the procedure. The procedure places no restriction on the initiation and termination points and the method therefore implicitly considers failure of the individual slopes as well as the multi-slope as a whole to find the global minimum factor of safety.

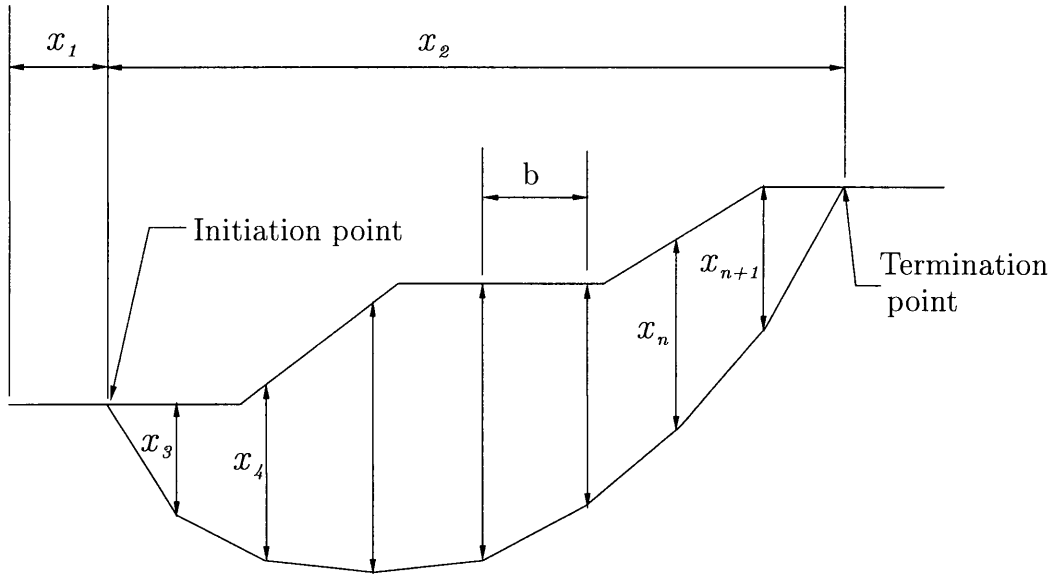

Figure 9.1: Definition of the geometric variables.

Chapter 10

Conclusions and Recommendations

10.1 Conclusions

The main objective of this study is the development of optimization methods to solve practical optimization problems in engineering. This goal is attained in the following ways:

- A probabilistic global stopping criterion, previously derived for a specific algorithm, is extended to multi-start algorithms, and is denoted the unified Bayesian stopping criterion. The suitability of the unified Bayesian stopping criterion is demonstrated for a number of local and global optimization algorithms using standard test functions.
- A multi-start global optimization infrastructure based on multiple local searches, combined with the unified Bayesian stopping criterion, is presented. Numerical results reveal that this simple multi-start approach outperforms a number of leading contenders.
- Parallelization of sequential multi-start algorithms is shown to effectively reduce the computational time associated with solving expensive global programming problems.
- Two algorithms simulating natural phenomena are implemented, namely the relatively new particle swarm optimization method and the well known genetic algorithm. For the current implementations, numerical results indicate that the computational effort associated with these methods are comparable.
- The observation that no single global optimization algorithm can consistently outperform all other algorithms when large sets of problems are considered, lead to the development of a parallel competing algorithm infrastructure. In this infrastructure different algorithms, ranging from deterministic to stochastic, compete simultaneously for a contribution to the unified Bayesian stopping criterion.
- The constrained global programming problem is addressed using constrained algorithms in the parallel competing algorithm infrastructure.

- An optimization procedure is developed for solving the slope stability problem faced in civil engineering. This new procedure determines the factor of safety of slopes using a global optimization approach.

10.2 Recommendations

1. The applicability of the Unified Bayesian stopping rule presented in Chapter 2 when used in combination with different algorithms should be investigated further.

2. An analysis of the effect of the Unified Bayesian stopping rule for functions for which the probability of convergence to the global optimum is less than the probability of convergence of some other local optimum is desirable.

3. Additional algorithms based on multiple local searches (Chapter 3) and the Unified Bayesian stopping rule can be formulated.

4. For both the successive genetic algorithm (Chapter 4) and the particle swarm algorithm (Chapter 5), additional operators can be formulated, in an attempt to improve the convergence characteristics of the algorithms.

5. In the competing algorithm infrastructure, additional algorithms should be incorporated. This is true for both the unconstrained and constrained infrastructures. A promising algorithm is the Lipschitzan DIRECT optimizer proposed by Jones *et al* [34].

   In addition, the algorithms in the infrastructure should be selected based on performance, as to exclude inefficient algorithms. Performance should be based on a larger test set than the set considered in this study.

6. The slope stability procedure presented in Chapter 9 can be improved in the following ways:

   (a) For single slope problems (see Examples 1 to 3), the starting values for the initiation point x_1 can randomly be selected close to the slope's toe and the termination point $(x_1 + x_2)$ close to the slope's crest.

   (b) Due to the difference in function evaluation time, it seems practical to perform the global optimization phase with Janbu's method and the local refinement phase with Spencer's method.

Bibliography

[1] C.A. Floudas and P.M. Pardalos. *A collection of Test Problems for Constrained Global Optimization Algorithms*, volume 455 of *Lecture notes in computer science*. Springer-Verlag, Berlin, Heidelberg, 1990.

[2] F. Schoen. Stochastic techniques for global optimization: A survey of recent advances. *J. Global Optim.*, 1:207–228, 1991.

[3] J.P. Lourens, H. Czapla, and A.L. Parrock. Finite element analysis of failure and structural rehabilitation of a high embankment on a soft foundation. *The Civil Engineer in South Africa*, pages 211–220, 1989.

[4] J.B. Lechman and D.V. Griffiths. Analysis of progressive failure of earth slopes by finite elements. In Geotechnics for Developing Africa. Wardle G.R. et al (eds.), editor, *Proceedings of the 12th African Regional Conference of the ISSMGE.*, pages 577–596, Balkema, Rotterdam., 1999.

[5] A. Törn and A. Zilinskas. *Global optimization*, volume 350 of *Lecture notes in computer science*. Springer-Verlag, Berlin, Heidelberg, 1989.

[6] C.G.E. Boender and A.H.G. Rinnooy Kan. A Bayesian analysis of the number of cells of a multinomial distribution,. *Statistician*, 32:240–248, 1983.

[7] J.A. Snyman and L.P. Fatti. A multi-start global minimization algorithm with dynamic search trajectories. *J. Optim. Theory Appl.*, 54:121–141, 1987.

[8] C.G.E. Boender, A.H.G. Rinnooy Kan, G.T. Timmer, and L. Stougie. A stochastic method for global optimization. *Math. Program.*, 22:125–140, 1982.

[9] A.H.G. Rinnooy Kan and G.T. Timmer. Stochastic global optimization methods, Part I: Clustering methods. *Math. Program.*, 39:27–56, 1987.

[10] R.H. Byrd, P. Lu, J. Nocedal, and C. Zhu. A limited memory algorithm for bound constrained optimization. *SIAM J. Scient. Comput.*, 16:1190–1208, 1995.

[11] C. Zhu, R.H. Byrd, P. Lu, and J. Nocedal. L-BFGS-B: FORTRAN subroutines for large scale bound constrained optimization. Technical Report NAM-11, Northwestern University, EECS Department, 1994.

[12] J.E. Dennis Jr. and J.J. Moré. Quasi-newton methods, motivation and theory. *SAIM Review*, 19, No. 1:46–89, 1977.

[13] J.C. Gilbert and J. Nocedal. Global convergence properties of conjugate gradient methods. *SIAM J. Optim.*, 2, 1992.

[14] J.A. Snyman and A.M. Hay. The spherical quadratic steepest descent (SQSD) method for unconstrained minimization with no explicit line searches. *Comp. Math. Appl.*, To appear, 2000.

[15] J. Mockus. *Bayesian Approach to Global Optimization.* Kluwer Academic Publishers, Dordrecht, Netherlands, 1989.

[16] J. Mockus. Application of bayesian approach to numerical methods of global and stochastic optimization. *J. Global Opt.*, 4:347–36, 1994.

[17] J.B. Lee and B.C. Lee. A global optimization algorithm based on the new filled function method and the genetic algorithm. *Eng. Opt.*, 27:1–20, 1996.

[18] A.A. Groenwold and J.A. Snyman. Global optimization using dynamic search trajectories. In *Proc. Conference Discrete and Global Optimization*, Chania, Crete, May 1998.

[19] F. Aluffi-Pentini, V. Parisi, and F. Zirilli. Global optimization and stochastic differential equations. *J. Optim. Theory Appl.*, 47:1–16, 1985.

[20] F. Aluffi-Pentini, V. Parisi, and F. Zirilli. SIGMA - a stochastic-integration global minimization algorithm. *ACM Trans. Math. Softw.*, 14:366–380, 1988.

[21] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine learning.* Addison-Wesley, MA, 1989.

[22] A.A. Groenwold, N. Stander, and J.A. Snyman. A regional genetic algorithm for the discrete optimal design of truss structures. *Int. J. Num. Meth. Eng.*, 44:749–766, 1999.

[23] A.T.C. Goh. Genetic algorithm search for critical slip surface in multiple-wedge stability analysis. *Can. Geotech. J.*, 36:382–391, 1999.

[24] D.L. Carroll. Chemical laser modelling with genetic algorithms. *AIAA Journal*, 34, No. 2:338–346, 1996.

[25] Y. Davidor. *Genetic algorithms and robotics: a heuristic strategy for optimization.* London, England, 1991.

[26] E. Potgieter. *A Genetic algorithm for the discrete structural optimization of laminated plates.* M. Eng. Dissertation, Department of Mechanical Engineering, University of Pretoria, 1997.

[27] C. Mattheck and S. Burkhardt. A new method of structural shape optimization based on biological growth. *Int. J. Fatigue*, 12:185–190, 1990.

[28] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equations of state calculations by fast computing machine. *J. Chem. Physics*, 21:1084–1092, 1953.

[29] J. Kennedy and Eberhart R. Particle swarm optimization. *proceedings of the IEEE International Conference on Neural Network*, pages 1942–1948, 1995.

[30] P.C. Fourie and A.A. Groenwold. Particle swarms in size and shape optimization. In J.A. Snyman and K. Craig, editors, *Proc. Workshop on Multidisciplinary Design Optimization*, pages 97–106, Pretoria, South Africa, August 2000.

[31] Y. Shi and R. Eberhart. Parameter selection in particle swarm optimization. *The Seventh Annual Conference on Evolutionary Programming*, 1998.

[32] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. PVM: Parallel Virtual Machine - A users guide and tutorial for networked parallel computing. ftp://netlib2.cs.utk.edu/pvm3/, 1997. (ver. 3.4).

[33] J.F. Schutte, H.P.J. Bolton, C. Erasmus, S. Geyer, and A.A. Groenwold. An efficient parallel global optimization infrastructure for composite structures. In *The Fifth International Conference on Computational Structures Technology*, Leuven, Belgium, Sept. 2000. Accepted.

[34] D.R. Jones, C.D. Pertunnen, and B.E. Stuckman. Lipschitzan optimization without the lipschitz constant. *J. Opt. Theory Appl.*, 79:157–181, 1993.

[35] Vanderplaats Research & Development, Inc. *DOT: Design Optimization Tools*, 1995. Version 4.2.

[36] J.A. Snyman. A new and dynamic method for unconstrained minimization. *Appl. Math. Modelling*, 6:449–462, 1982.

[37] J.A. Snyman. An improved version of the original leap-frog dynamic method for unconstrained minimization: LFOP1(b). *Appl. Math. Modelling*, 7:216–218, 1983.

[38] J.A. Snyman. The LFOPC leap-frog method for constrained optimization. *Comp. Math. Appl.*, To appear, 2000.

[39] S.E. Cox, R.T. Haftka, C.A. Baker, B. Grossman, W.H. Mason, and L.T. Watson. Global optimization for noise and multiple local optima. In J.A. Snyman and K. Craig, editors, *Proc. Workshop on Multidisciplinary Design Optimization*, pages 50–59, Pretoria, South Africa, August 2000.

[40] J.A. Snyman, W.J. Roux, and Stander N. A dynamic penalty function method for the solution of structural optimization problems. *Appl. Math. Modelling*, 18:453–460, 1994.

[41] J.A. Snyman and A.M. Hay. The Dynamic-Q optimization method: An alternative to SQP. *Proceedings of the International Workshop on multi disciplinary design optimization*, pages 163–172, 2000.

[42] A.V. Fiacco and G.P. McCormick. *Nonlinear Programming: Sequential Unconstrained Minimization Techniques.* John Wiley and Sons, New York, 1968.

[43] J.A. Snyman. Unconstrained minimization by combining the dynamic and conjugate gradient methods. *Quaestiones Mathematicae*, 8:33–42, 1985.

[44] N. Janbu. Stability analysis of slopes with dimensionless parameters. *Harvard Soil Mechanics Series*, 46, 1954.

[45] N. Janbu. Earth pressure and bearing capacity calculations by generalized procedure of slices. *Proceedings of the 4th International Conference of the ISSMFE*, 2:207–212, 1957.

[46] N.R. Morgenstern and V.E. Price. The analysis of the stability of general slip surfaces. *Geotechnical and Geological Engineering*, 1:79–93, 1965.

[47] E. Spencer. A method of analysis for stability of embankments using parallel inter-slice forces. *Geotechnique*, 17:11–26, 1967.

[48] E. Spencer. Thrust line criterion in embankment stability analysis. *Geotechnique*, 23:85–100, 1973.

[49] R.A. Siegel, W.D. Kovacs, and C.W. Lovell. Random search generation in stability analysis. *ASCE*, 107, No. GT7:996–1002, 1981.

[50] V.U. Nguyen. Determination of critical slope failure surfaces. *J. Geotech. Eng.*, 111, No. 2:238–250, 1985.

[51] T.B. Celestino and J.M. Duncan. Simplified search for non-circular slip surface. *Poceedings of the 10th International Conference on Soil Mechanics and Foundation Engineering*, 1981.

[52] K.S. Li and W. White. Rapid evaluation of the critical slip surface in slope stability problems. *Int. J. Num. Anal. Meth. Geomech.*, 11:449–473, 1987.

[53] R. Baker. Determination of critical slip surface in slope stability computations. *Int. J. Num. Anal. Meth. Geomech.*, 4:333–359, 1980.

[54] D.G. Fredlund and J. Krahn. Comparison of slope stability methods of analysis. *Can. Geotech. J.*, 14:429–439, 1977.

[55] F. Wagner. The merriespruit slimes dam failure: Overview and lessons learnt. *SAICE Journal*, 39:11–15, 1997.

[56] N. Janbu, L. Bjerrum, and B. Kjaernsli. Veilidning ved losning av fundamenteringsoppgraver (soil mechanics applied to some engineering problems). *N.G.I. Publication*, 16, 1956.

[57] C.R.I. Clayton, J. Milititsky, and R.I. Woods. *Earth pressure and earth retaining structures.* 2nd Edition. Blackie Acadamic and Professional, England, London, 1993.

[58] W. Hock and K. Schittkowski. *Test examples for nonlinear programming codes*, volume 187 of *Lecture notes in Economics and Mathematical Systems*. Springer-Verlag, Berlin, Heidelberg, New York, 1981.

Appendix A

The extended Dixon-Szegö unconstrained test set

Problems 1 and 2 (Griewank G1 and G2 functions, respectively)
OBJECTIVE FUNCTION:

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} x_i^2/d - \prod_{i=1}^{n} \cos\left(x_i/\sqrt{i}\right) + 1.$$

For Problem 1, $n = 2$ and $d = 200$; for Problem 2, $n = 10$ and $d = 4000$.

SEARCH DOMAIN FOR 1:

$$D = \{(x_1, x_2) \in R^2 : -100.0 \leq x_i \leq 100.0, \quad i = 1, 2\}.$$

SEARCH DOMAIN FOR 2:

$$D = \{(x_1, x_2, \cdots, x_{10}) \in R^{10} : -600.0 \leq x_i \leq 600.0, \quad i = 1, 2, \cdots, 10\}.$$

SOLUTION:

$$\boldsymbol{x}^* = (0.0, \cdots, 0.0) \quad f^* = 0.0.$$

Problem 3 (Goldstein-Price)
OBJECTIVE FUNCTION:

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

SEARCH DOMAIN:

$$D = \{(x_1, x_2) \in R^2 : -2.0 \leq x_i \leq 2.0, \quad i = 1, 2\}.$$

SOLUTION:

$$\boldsymbol{x}^* = (0.0, -1.0) \quad f^* = 3.0.$$

Problem 4 (Six-hump Camelback)

OBJECTIVE FUNCTION:

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

SEARCH DOMAIN:

$$D = \{x_1 \in R^1 : -3.0 \leq x_1 \leq 3.0\}$$
$$D = \{x_2 \in R^1 : -2.0 \leq x_2 \leq 2.0\}$$

SOLUTION:

$$\boldsymbol{x}_1^* = (0.0898, -0.7126) \quad \boldsymbol{x}_2^* = (-0.0898, 0.7126) \quad f^* = -1.0316285$$

Problem 5 (Shubert function, Levi no. 4)

OBJECTIVE FUNCTION:

$$f(\boldsymbol{x}) = \{\sum_{i=1}^{5} i\cos[(i+1)x_1 + i]\}\{\sum_{i=1}^{5} i\cos[(i+1)x_2 + i]\}$$

SEARCH DOMAIN:

$$D = \{(x_1, x_2) \in R^2 : -10.0 \leq x_i \leq 10.0, \quad i = 1, 2\}$$

SOLUTION:

$$\boldsymbol{x}_1^* = (5.48289, -1.426531) \quad f^* = -186.73091$$

Problem 6 (Rastrigin)

OBJECTIVE FUNCTION:

$$f(\boldsymbol{x}) = x_1^2 + x_2^2 - \cos(18x_1) - \cos(18x_2)$$

SEARCH DOMAIN:

$$D = \{(x_1, x_2) \in R^2 : -1.0 \leq x_i \leq 1.0, \quad i = 1, 2\}.$$

SOLUTION:

$$\boldsymbol{x}^* = (0.0, 0.0) \quad f^* = -2.0$$

Problem 7 (Branin)

OBJECTIVE FUNCTION:

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

SEARCH DOMAIN:

$$D = \{x_1 \in R^1 : -5.0 \leq x_1 \leq 10.0\}$$

$$D = \{x_2 \in R^1 : 0.0 \leq x_2 \leq 15.0\}$$

SOLUTION:

$$\boldsymbol{x}_1^* \approx (3.142, 2.275) \quad f^* \approx 0.398$$

Problem 8 and 9 (Hartman 3, 6)

OBJECTIVE FUNCTION:

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

where $x = (x_1, \ldots, x_n)$, and

H3 : $m = 4,\ n = 3$

i	a_{ij}			c_i	p_{ij}		
1	3.0	10.0	30.0	1.0	0.3689	0.1170	0.2673
2	0.1	10.0	35.0	1.2	0.4699	0.4387	0.7470
3	3.0	10.0	30.0	3.0	0.1091	0.8732	0.5547
4	0.1	10.0	35.0	3.2	0.03815	0.5743	0.8828

H6 : $m = 4,\ n = 6$

i	a_{ij}						c_i
1	10.0	3.0	17.0	3.5	1.7	8.0	1.0
2	0.05	10.0	17.0	0.1	8.0	14.0	1.2
3	3.0	3.5	1.7	10.0	17.0	8.0	3.0
4	17.0	8.0	0.05	10.0	0.1	14.0	3.2

i	p_{ij}					
1	0.1312	0.1696	0.5569	0.0124	0.8283	0.5886
2	0.2329	0.4135	0.8307	0.3736	0.1004	0.9991
3	0.2348	0.1451	0.3522	0.2883	0.3047	0.6650
4	0.4047	0.8828	0.8732	0.5743	0.1091	0.0381

SEARCH DOMAIN:

$$D = \{(x_1, \ldots, x_n) \in R^n : 0.0 \leq x_i \leq 1.0,\ \ i = 1, \ldots, n\}$$

SOLUTIONS:

H3:

$$x^* = (0.11461478, 0.55564892, 0.85254688), \quad f^* = -3.8627821.$$

H6:

$$x^* = (0.20168955, 0.15000963, 0.47687211, 0.27533377, 0.31165102, 0.65730111),$$

$$f^* = -3.322368.$$

Problem 10, 11 and 12 (Shekel 5, 7, 10)

OBJECTIVE FUNCTION:

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

where:

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

SEARCH DOMAIN:

$$D = \{(x_1, \ldots, x_4) \in R^4 : 0.0 \leq x_i \leq 10.0,\ i = 1, \ldots, 4\}.$$

SOLUTIONS:

S5:

$$x^* = (4.00003727, 4.00013375, 4.00003730, 4.00013346)\ \ f^* = -10.153200.$$

S7:

$$x^* = (4.00057280, 4.00069020, 3.99948997, 3.99960620)\ \ f^* = -10.402941.$$

S10:

$$x^* = (4.00074671, 4.00059326, 3.99966290, 3.99950981)\ \ f^* = -10.536410.$$

Appendix B

The constrained test set

Constrained Problem 1 (C1)

OBJECTIVE FUNCTION:

$$f(\boldsymbol{x}) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

CONSTRAINTS:

$$g_1(\boldsymbol{x}) = -(x_2^2 - x_1)$$
$$g_2(\boldsymbol{x}) = -(x_1^2 - x_2)$$

SEARCH DOMAIN:

$$x_1 \in R^1 : -0.5 \leq x_1 \leq 0.5$$
$$x_2 \in R^1 : -20.0 \leq x_2 \leq 1.0$$

SOLUTION:

$$\boldsymbol{x}_1^* = (0.0, 0.0) \quad f^* = 1.0$$

Constrained Problem 2 (C2)

OBJECTIVE FUNCTION:

$$f(\boldsymbol{x}) = 7x_1 + x_2 + 3x_3 + x_4$$

CONSTRAINTS:

$$g_1(\boldsymbol{x}) = 8 - (2x_1 - 3x_2 - x_3 + x_4)$$
$$g_2(\boldsymbol{x}) = 12 - (6x_1 + x_2 + 2x_3 - 2x_4)$$
$$g_3(\boldsymbol{x}) = 10 + (x_1 - x_2 - x_3 - x_4)$$

SEARCH DOMAIN:

$$(x_1, x_2, x_3, x_4) \in R^4 : 0.0 \leq x_i \leq 20.0, \quad i = 1, 2, 3, 4.$$

SOLUTION:

$$\boldsymbol{x}_1^* = (2.8, 0.0, 5.2, 7.6) \quad f^* = 42.8$$

Constrained Problem 3 (C3)

OBJECTIVE FUNCTION:

$$f(\boldsymbol{x}) = x_1^3 - 6x_1^2 + 11x_1 + x_3$$

CONSTRAINTS:

$$g_1(\boldsymbol{x}) = x_1^2 + x_2^2 - x_3^2$$
$$g_2(\boldsymbol{x}) = 4 - (x_1^2 + x_2^2 + x_3^2)$$

SEARCH DOMAIN:

$$x_1 \in R^1 : 0.0 \leq x_1 \leq 20.0$$
$$x_2 \in R^1 : 0.0 \leq x_2 \leq 20.0$$
$$x_3 \in R^1 : 0.0 \leq x_3 \leq 5.0$$

SOLUTION:

$$\boldsymbol{x}_1^* = (0.0, 1.4142, 1.4142) \quad f^* = 1.4142$$

Constrained Problem 4 (C4)

OBJECTIVE FUNCTION:

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

CONSTRAINTS:

$$g_1(\boldsymbol{x}) = x_1 + x_2 - 2$$
$$g_2(\boldsymbol{x}) = x_1^2 - x_2$$

SEARCH DOMAIN:

$$x_1 \in R^1 : -20.0 \leq x_1 \leq 20.0$$
$$x_2 \in R^1 : -20.0 \leq x_2 \leq 20.0$$

SOLUTION:

$$\boldsymbol{x}_1^* = (1.0, 1.0) \quad f^* = 1.0$$

Constrained Problem 5 (C5)

OBJECTIVE FUNCTION:

$$f(\boldsymbol{x}) = 2 - \frac{1}{120} x_1 x_2 x_3 x_4 x_5$$

SEARCH DOMAIN:

$$x_1 \in R^1 : 0.0 \leq x_1 \leq 1.0$$
$$x_2 \in R^1 : 0.0 \leq x_2 \leq 2.0$$
$$x_3 \in R^1 : 0.0 \leq x_2 \leq 3.0$$
$$x_4 \in R^1 : 0.0 \leq x_2 \leq 4.0$$
$$x_5 \in R^1 : 0.0 \leq x_2 \leq 5.0$$

SOLUTION:

$$\boldsymbol{x}_1^* = (1.0, 2.0, 3.0, 4.0, 5.0) \quad f^* = 1.0$$

Constrained Problem 6 [58] (C6)

OBJECTIVE FUNCTION:

$$f(\boldsymbol{x}) = -x_1 x_2 x_3$$

CONSTRAINTS:

$$h_1(\boldsymbol{x}) = x_1 - 4.2(\sin x_4)^2 = 0$$
$$h_2(\boldsymbol{x}) = x_2 - 4.2(\sin x_5)^2 = 0$$
$$h_3(\boldsymbol{x}) = x_3 - 4.2(\sin x_6)^2 = 0$$
$$h_4(\boldsymbol{x}) = x_1 + 2x_2 + 2x_3 - 7.2(\sin x_7)^2 = 0$$

SEARCH DOMAIN:

$$(x_1, x_2, \cdots, x_7) \in R^7 : -20.0 \leq x_i \leq 20.0, \quad i = 1, 2, \cdots, 7.$$

SOLUTION:

$$\boldsymbol{x}_1^* = (2.4, 1.2, 1.2, *, *, *, *) \quad f^* = -3.4560$$

Constrained Problem 7 [58] (C7)

OBJECTIVE FUNCTION:

$$f(\boldsymbol{x}) = (x_1 - 10)^2 + 5(x_2 - 12)^2 + x_3^4 + 3(x_4 - 11)^2 + 10x_5^6 + 7x_6^2 + x_7^4 - 4x_6 x_7 - 10x_6 - 8x_7$$

CONSTRAINTS:

$$g_1(\boldsymbol{x}) = 2x_1^2 + 3x_2^4 + x_3 + 4x4^2 + 5x_5 - 127 \leq 0$$
$$g_2(\boldsymbol{x}) = 7x_1 + 3x_2 + 10x_3^2 + x4 - x_5 - 282 \leq 0$$
$$g_3(\boldsymbol{x}) = 2x_1^2 + 3x_2^4 + x_6 + 4x7^2 - 196 \leq 0$$
$$g_4(\boldsymbol{x}) = 4x_1^2 + x_2^2 - 3x_1 x2 + 2x_3^2 + 5x_6 - 11x_7 \leq 0$$

SEARCH DOMAIN:

$$(x_1, x_2, \cdots, x_7) \in R^7 : -20.0 \leq x_i \leq 20.0, \quad i = 1, 2, \cdots, 7.$$

SOLUTION:

$$\boldsymbol{x}_1^* = (2.3305, 1.9513, -0.47754, 4.3658, -0.62448, 1.0381, 1.5942) \quad f^* = 680.63$$

Appendix C

Proof of stopping criterion

An outline of the proof of (2.4) is presented, which follows closely the proof in [7].

Given $\tilde{n}^*$ and α^*, the probability that at least one point, $\tilde{n} \geq 1$, has converged to f^* is

$$\Pr[\tilde{n}^* \geq 1|\tilde{n}, r] = 1 - (1 - \alpha^*)^{\tilde{n}} \; . \tag{C.1}$$

In the Bayesian approach, the uncertainty about the value of α^* is characterized by specifying a prior probability distribution for it. This distribution is modified using the sample information (namely, $\tilde{n}$ and r) to form a posterior probability distribution. Let $p_*(\alpha^*|\tilde{n}, r)$ be the posterior probability distribution of α^*. Then,

$$\begin{aligned} \Pr[\tilde{n}^* \geq 1|\tilde{n}, r] &= \int_0^1 \left[1 - (1 - \alpha^*)^{\tilde{n}}\right] p_*(\alpha^*|\tilde{n}, r) d\alpha^* \\ &= 1 - \int_0^1 (1 - \alpha^*)^{\tilde{n}} p_*(\alpha^*|\tilde{n}, r) d\alpha^*. \end{aligned} \tag{C.2}$$

Now, although the r sample points converge to the current overall minimum, it is not known whether this minimum corresponds to the global minimum of f^*. Utilizing (2.3), and noting that $(1 - \alpha)^{\tilde{n}}$ is a decreasing function of α, the replacement of α^* in the above integral by α yields

$$\Pr[\tilde{n}^* \geq 1|\tilde{n}, r] \geq \int_0^1 \left[1 - (1 - \alpha)^{\tilde{n}}\right] p(\alpha|\tilde{n}, r) d\alpha \; . \tag{C.3}$$

Now, using Bayes theorem:

$$p(\alpha|\tilde{n}, r) = \frac{p(r|\alpha, \tilde{n}) p(\alpha)}{\int_0^1 p(r|\alpha, \tilde{n}) p(\alpha) d\alpha} \; . \tag{C.4}$$

Since the $\tilde{n}$ points are sampled at random and each point has a probability α of converging to the current overall minimum, r has a binomial distribution with parameters α and $\tilde{n}$. Therefore

$$p(r|\alpha, \tilde{n}) = \binom{\tilde{n}}{r} \alpha^r (1 - \alpha)^{\tilde{n} - r} \; . \tag{C.5}$$

Substituting (C.5) and (C.4) into (C.3) gives:

$$\Pr[\tilde{n}^* \geq 1|\tilde{n}, r] \geq 1 - \frac{\int_0^1 \alpha^r (1 - \alpha)^{2\tilde{n} - r} p(\alpha) d\alpha}{\int_0^1 \alpha^r (1 - \alpha)^{\tilde{n} - r} p(\alpha) d\alpha} \; . \tag{C.6}$$

A suitable flexible prior distribution $p(\alpha)$ for α is the beta distribution with parameters a and b:

$$p(\alpha) = [1/\beta(a,b)]\,\alpha^{a-1}(1-\alpha)^{b-1}, \qquad 0 \leq \alpha \leq 1 \tag{C.7}$$

Using this prior distribution gives:

$$\begin{aligned}\Pr[\tilde{n}^* \geq 1|\tilde{n},r] \;&\geq\; 1 - \frac{\Gamma(\tilde{n}+a+b)\;\Gamma(2\tilde{n}-r+b)}{\Gamma(2\tilde{n}+a+b)\;\Gamma(\tilde{n}-r+b)} \\ &=\; 1 - \frac{(\tilde{n}+a+b-1)!\;(2\tilde{n}-r+b-1)!}{(2\tilde{n}+a+b-1)!\;(\tilde{n}-r+b-1)!},\end{aligned}$$

which is the required result.

Appendix D

Slope geometries for examples and critical failure plane figures

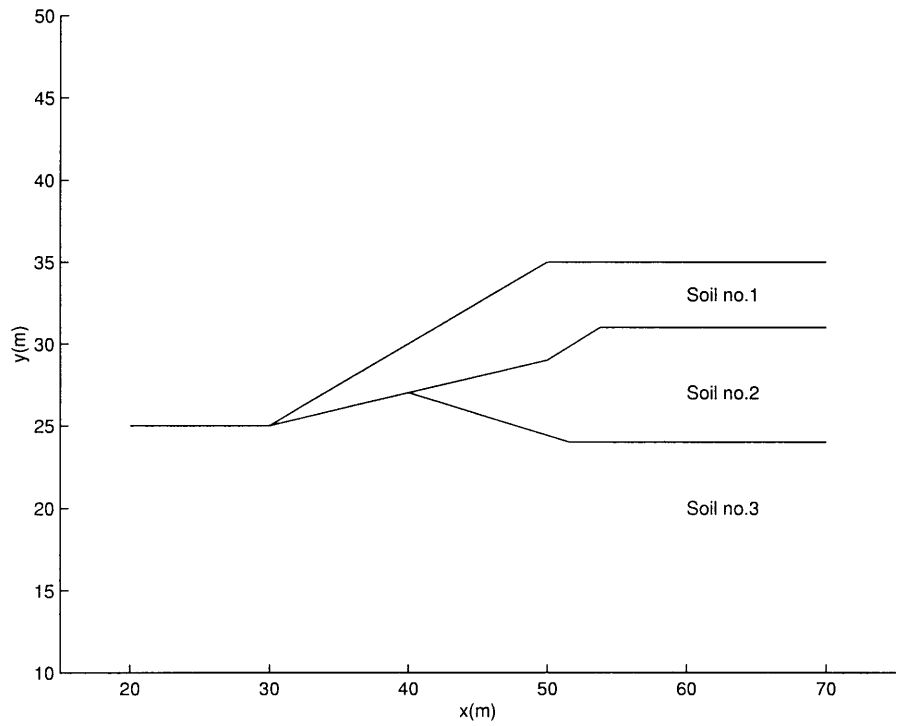

Figure D.1: Slope geometry for Example 1.

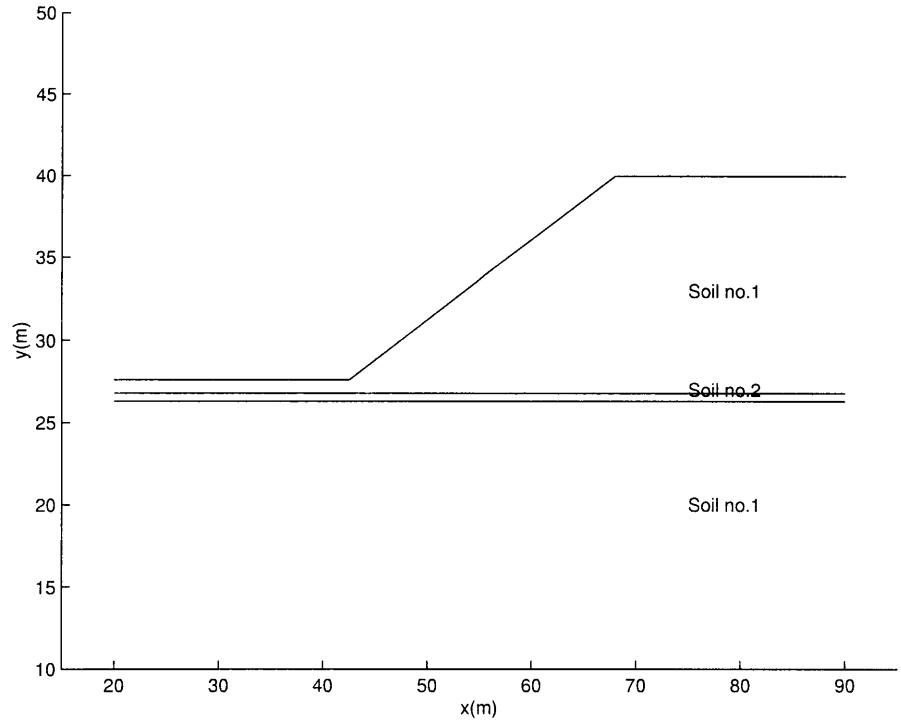

Figure D.2: Slope geometry for Example 2.

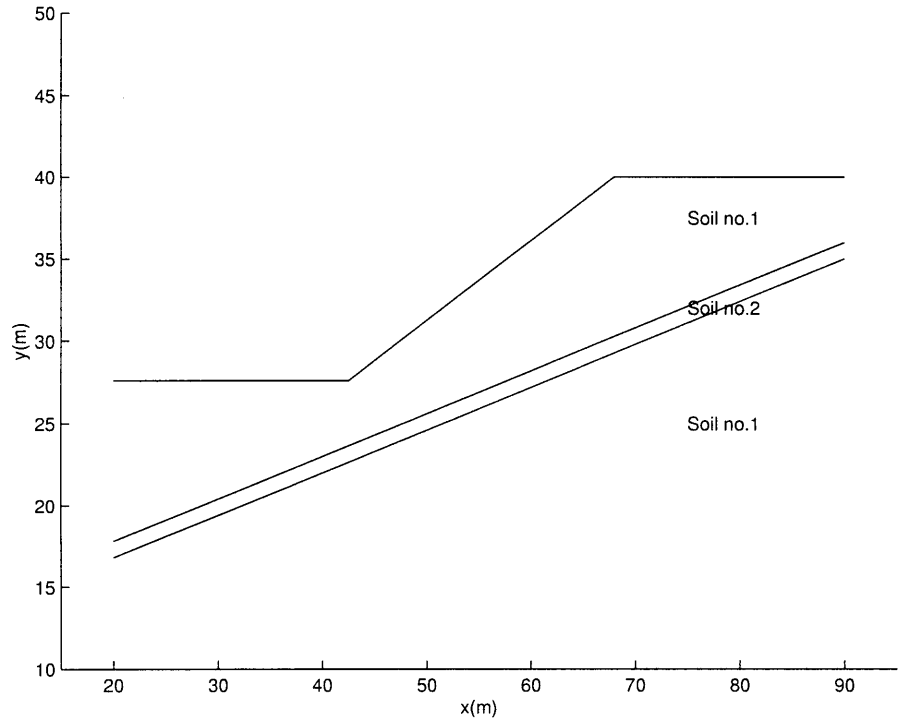

Figure D.3: Slope geometry for Example 3.

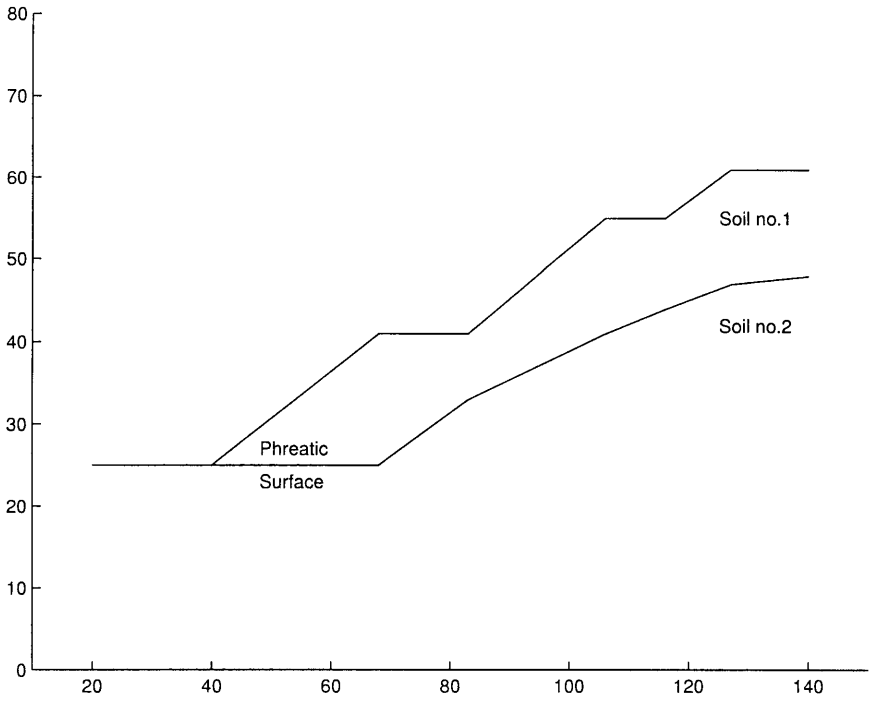

Figure D.4: Slope geometry for Example 4.

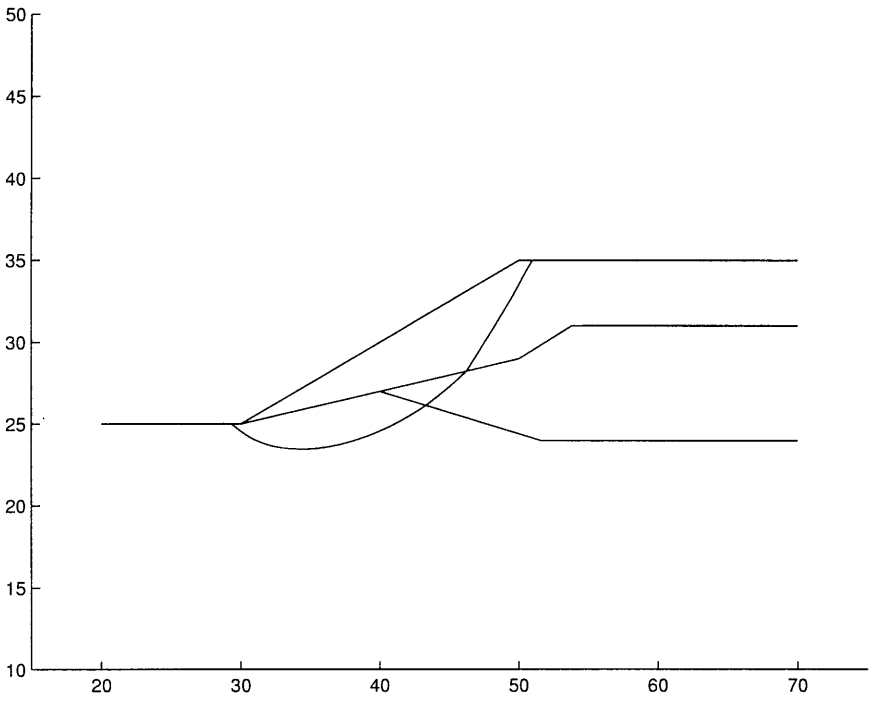

Figure D.5: Critical Failure plane found by the Leapfrog-Janbu analysis for Example 1.

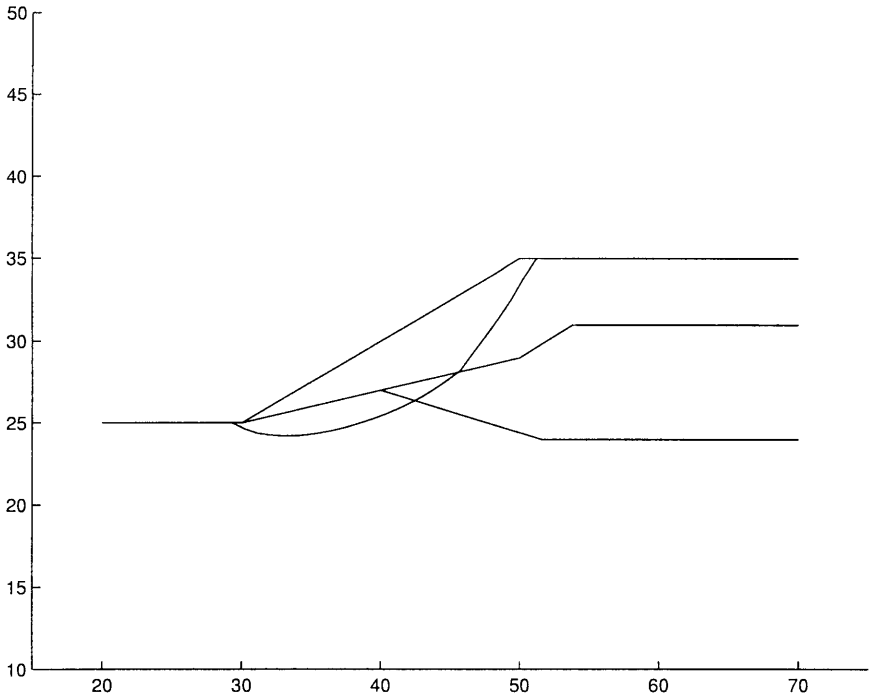

Figure D.6: Critical Failure plane found by the Leapfrog-Spencer analysis for Example 1.

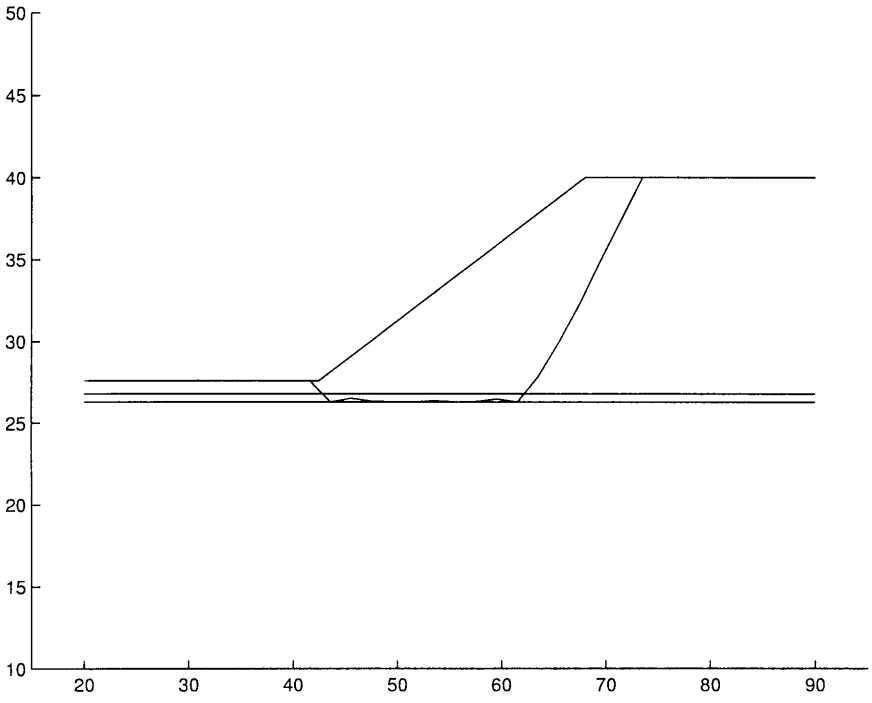

Figure D.7: Critical Failure plane found by the Leapfrog-Janbu analysis for Example 2.

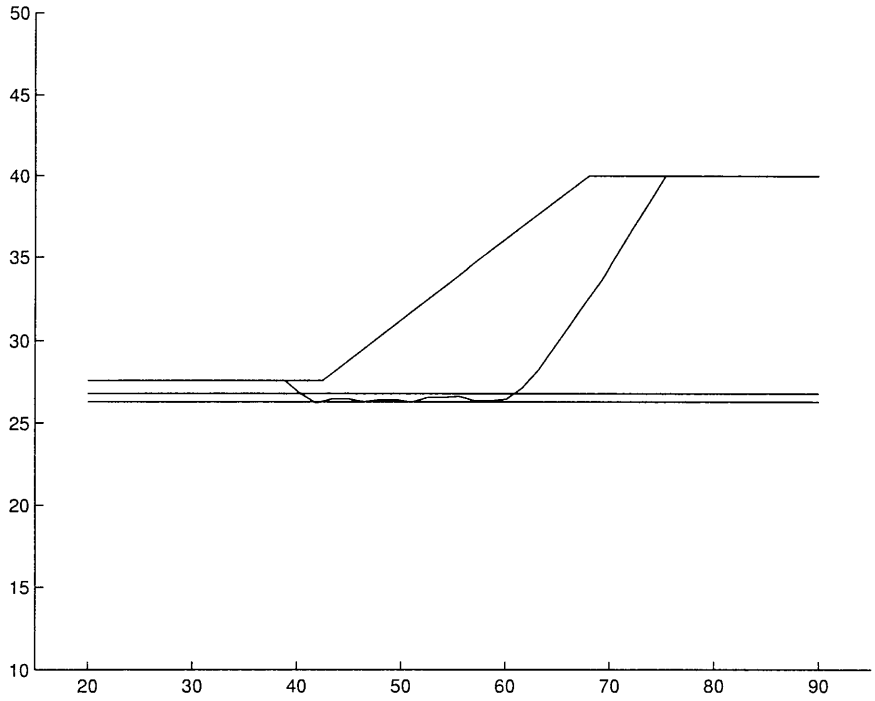

Figure D.8: Critical Failure plane found by the Leapfrog-Spencer analysis for Example 2.

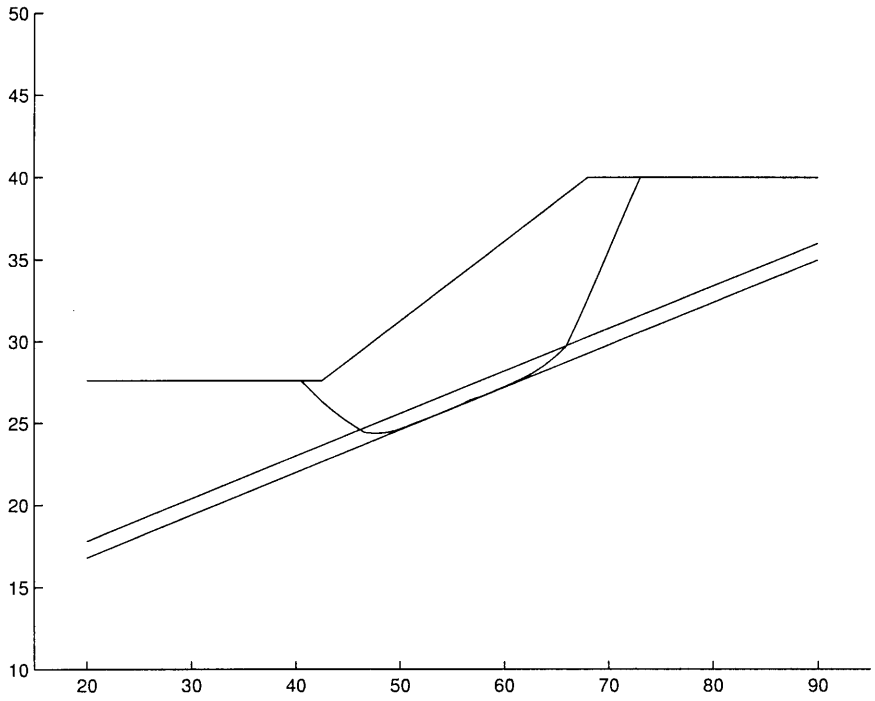

Figure D.9: Critical Failure plane found by the Leapfrog-Janbu analysis for Example 3.

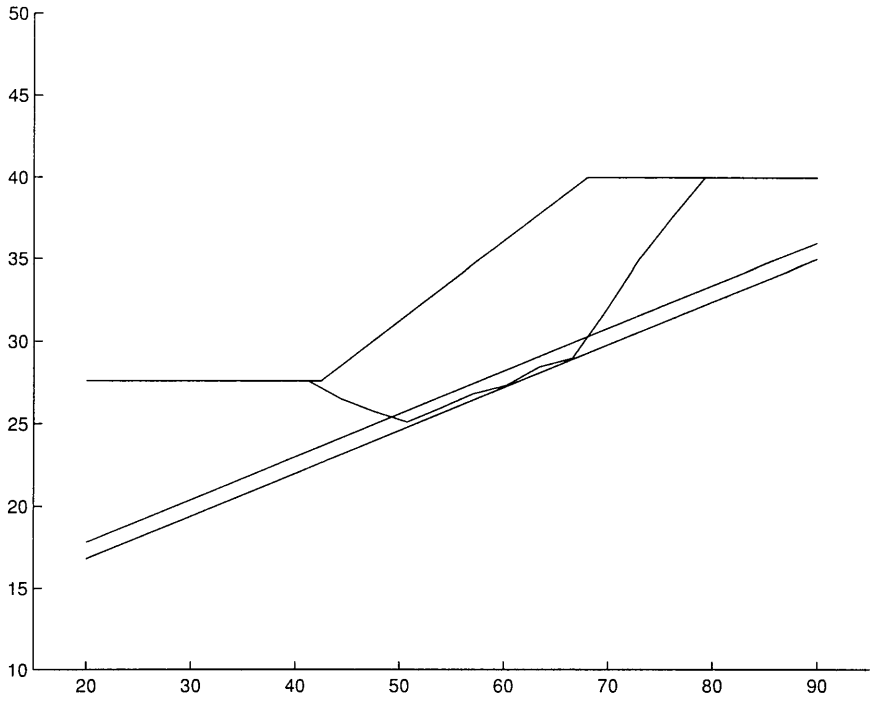

Figure D.10: Critical Failure plane found by the Leapfrog-Spencer analysis for Example 3.

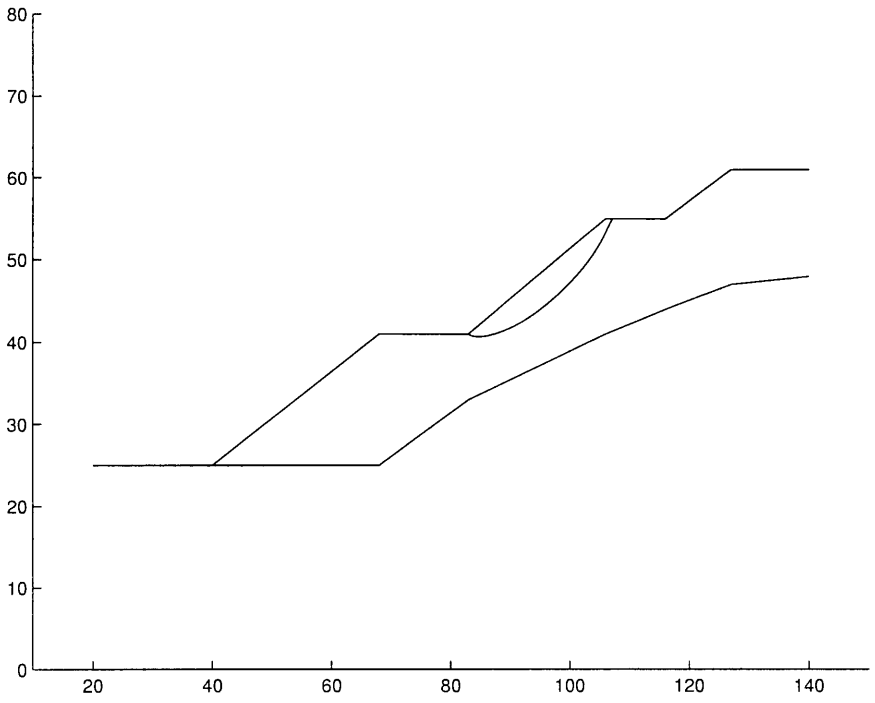

Figure D.11: Critical Failure plane found by the Leapfrog-Janbu analysis for Example 4.

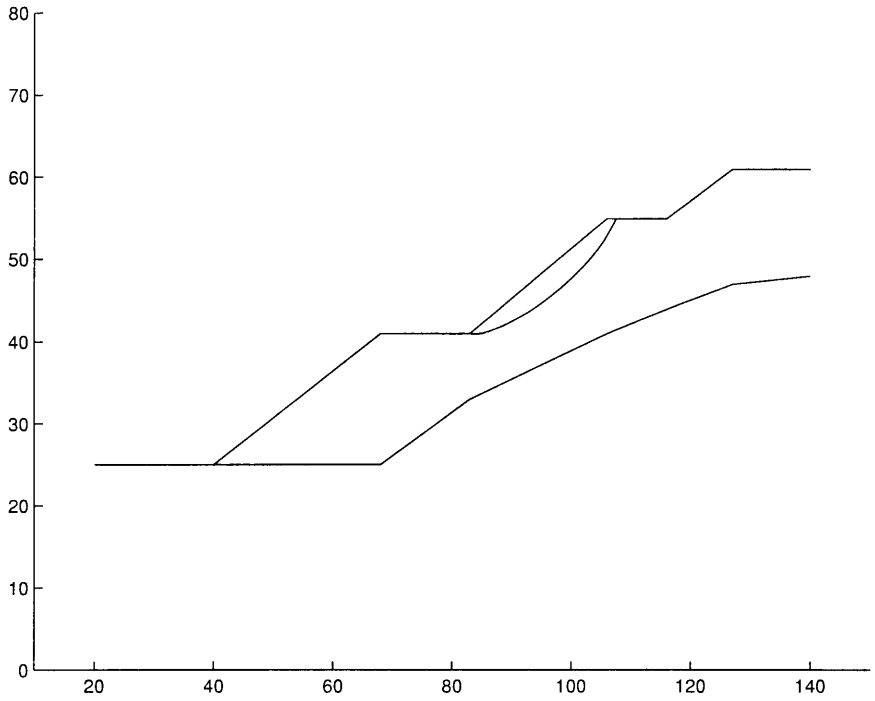

Figure D.12: Critical Failure plane found by the Leapfrog-Spencer analysis for Example 4.

Appendix E

Coordinates of critical failure planes

Table E.1: Critical failure plane coordinates calculated with Leapfrog-Janbu.

	Example. 1		Example. 2		Example. 3		Example. 4	
Node	x-coord.	y-coord.	x-coord.	y-coord.	x-coord.	y-coord.	x-coord.	y-coord.
1	29.323	25.000	41.599	27.600	40.480	27.600	82.999	41.000
2	29.661	24.786	43.593	26.304	41.498	26.967	83.376	40.849
3	30.000	24.559	45.588	26.520	42.515	26.323	83.753	40.759
4	30.338	24.356	47.582	26.348	43.533	25.783	84.130	40.706
5	30.677	24.180	49.576	26.310	44.551	25.288	84.508	40.683
6	31.015	24.035	51.570	26.295	45.568	24.852	84.885	40.679
7	31.353	23.909	53.564	26.382	46.586	24.445	85.262	40.696
8	31.692	23.802	55.559	26.293	47.604	24.375	85.639	40.725
9	32.030	23.716	57.553	26.326	48.621	24.428	86.016	40.766
10	32.369	23.643	59.547	26.489	49.639	24.577	86.393	40.824
11	32.707	23.589	61.541	26.296	50.657	24.809	86.770	40.885
12	33.046	23.546	63.535	27.859	51.674	25.076	87.147	40.956
13	33.384	23.513	65.529	29.982	52.692	25.316	87.524	41.038
14	33.722	23.491	67.524	32.318	53.710	25.577	87.901	41.127
15	34.061	23.477	69.518	34.967	54.727	25.847	88.278	41.223
16	34.399	23.473	71.512	37.486	55.745	26.109	88.655	41.325
17	34.738	23.478	73.506	40.000	56.762	26.445	89.032	41.433
18	35.076	23.492	—	—	57.780	26.639	89.410	41.547
19	35.415	23.515			58.798	26.910	89.787	41.668
20	35.753	23.546			59.815	27.185	90.164	41.795
21	36.091	23.586			60.833	27.469	90.541	41.928
22	36.430	23.633			61.851	27.779	90.918	42.068
23	36.768	23.689			62.868	28.137	91.295	42.212
24	37.107	23.751			63.886	28.567	91.672	42.365
25	37.445	23.822			64.904	29.092	92.049	42.522
26	37.784	23.898			65.921	29.710	92.426	42.686
27	38.122	23.983			66.939	31.081	92.803	42.855
28	38.460	24.073			67.957	32.527	93.180	43.030
29	38.799	24.171			68.974	34.014	93.557	43.210

Table E.1 continued:

Node	Example. 1		Example. 2		Example. 3		Example. 4	
	x-coord.	y-coord.	x-coord.	y-coord.	x-coord.	y-coord.	x-coord.	y-coord.
30	39.137	24.275	—	—	69.992	35.527	93.935	43.397
31	39.476	24.386			71.010	37.043	94.312	43.588
32	39.814	24.503			72.027	38.552	94.689	43.785
33	40.153	24.628			73.045	40.000	95.066	43.988
34	40.491	24.759			—	—	95.443	44.195
35	40.829	24.898					95.820	44.408
36	41.168	25.044					96.197	44.626
37	41.506	25.196					96.574	44.849
38	41.845	25.356					96.951	45.079
39	42.183	25.523					97.328	45.314
40	42.522	25.697					97.705	45.555
41	42.860	25.880					98.082	45.802
42	43.198	26.069					98.459	46.057
43	43.537	26.280					98.837	46.317
44	43.875	26.503					99.214	46.584
45	44.214	26.732					99.591	46.858
46	44.552	26.966					99.968	47.140
47	44.891	27.208					100.345	47.430
48	45.229	27.456					100.722	47.726
49	45.567	27.712					101.099	48.033
50	45.906	27.974					101.476	48.346
51	46.244	28.248					101.853	48.670
52	46.583	28.699					102.230	49.004
53	46.921	29.158					102.607	49.347
54	47.260	29.620					102.984	49.703
55	47.598	30.082					103.362	50.072
56	47.936	30.541					103.739	50.456
57	48.275	31.001					104.116	50.860
58	48.613	31.468					104.493	51.286
59	48.952	31.943					104.870	51.737
60	49.290	32.426					105.247	52.221
61	49.628	32.924					105.624	52.744
62	49.967	33.448					106.001	53.318
63	50.305	33.997					106.378	53.906
64	50.644	34.510					106.755	54.488
65	50.982	35.000					107.132	55.000

Table E.2: Critical failure plane coordinates calculated with Leapfrog-Spencer.

	Example. 1		Example. 2		Example. 3		Example. 4	
Node	x-coord.	y-coord.	x-coord.	y-coord.	x-coord.	y-coord.	x-coord.	y-coord.
1	29.333	25.000	38.836	27.600	41.211	27.600	83.000	41.000
2	30.245	24.620	40.357	26.806	44.383	26.494	84.023	40.945
3	31.157	24.365	41.879	26.223	47.554	25.742	85.046	41.068
4	32.069	24.251	43.400	26.485	50.726	25.101	86.069	41.271
5	32.981	24.204	44.922	26.483	53.898	25.935	87.092	41.529
6	33.893	24.216	46.444	26.302	57.069	26.817	88.115	41.825
7	34.805	24.291	47.965	26.417	60.241	27.332	89.138	42.164
8	35.717	24.407	49.487	26.439	63.412	28.460	90.161	42.544
9	36.629	24.555	51.008	26.266	66.584	28.982	91.185	42.951
10	37.541	24.747	52.530	26.566	69.756	31.829	92.208	43.380
11	38.453	24.978	54.051	26.569	72.927	34.941	93.231	43.850
12	39.365	25.233	55.573	26.635	76.099	37.535	94.254	44.343
13	40.277	25.519	57.095	26.373	79.271	40.000	95.277	44.855
14	41.189	25.842	58.616	26.367	—	—	96.300	45.414
15	42.101	26.208	60.138	26.466			97.323	46.003
16	43.013	26.618	61.659	27.160			98.346	46.615
17	43.925	27.112	63.181	28.243			99.370	47.263
18	44.837	27.643	64.703	29.600			100.393	47.963
19	45.749	28.227	66.224	30.970			101.416	48.695
20	46.661	29.295	67.746	32.360			102.439	49.473
21	47.573	30.322	69.267	33.695			103.462	50.335
22	48.485	31.407	70.789	35.253			104.485	51.272
23	49.397	32.541	72.311	36.861			105.508	52.309
24	50.309	33.870	73.832	38.409			106.531	53.620
25	51.221	35.000	75.354	40.000			107.554	55.000

Appendix F

Program Listings

F.1 Genetic Algorithm

```
C ********************************************************************** C
C ********************************************************************** C
C                                                                        C
C                    GENETIC ALGORITHM BY H.P.J.BOLTON                   C
C                                                                        C
C ********************************************************************** C
C ********************************************************************** C
      SUBROUTINE GA(F,N,X,BL,BU,NI,NE
     &,PFB,PFE,NBIN,NPOP,NGEN,BM,EM,FM,nSTIPE,
     #CP,NTOUR,DELTAXDELING,EINDGRENSGROOOTE,NVSK)
      IMPLICIT REAL*8(A-H,O-Z),INTEGER(I-N)
      INCLUDE 'params.inc'
      INTEGER POP(MAXPOP,MAXDNA*MAXSIZ),PPOP(MAXPOP,MAXDNA*MAXSIZ)
      INTEGER MNR(2),DNA,VOLGORDE(MAXPOP),OPTMETHOD,HERHAAL,RY,RYNR
      DOUBLE PRECISION GRENS(MAXSIZ,2),PERS(MAXPOP),FUNK(MAXPOP,2)
      DOUBLE PRECISION MAKSMUTC,MUT,FC(MAXPOP),GEBIED,FMINWAR
      DOUBLE PRECISION P,MUTW,XX(MAXPOP,MAXSIZ),TOURF(MAXSIZ,2)
      DOUBLE PRECISION FF(MAXPOP,2),DELTAX(MAXSIZ)
      DOUBLE PRECISION C(MAXCON),H(MAXCON),MAXDX,XMIN(MAXSIZ)
      DIMENSION X(N),BL(N),BU(N)
      COMMON /GAFEVAL/ NGAFEVAL
C
C *************************** INPUT PARAMETERS ********************** C
      PFB=1.0
      PFE=15.0
      NBIN = 10                != Number of binary bits per variable
      NPOP = 8                 != Number of population size
      NGEN = 30                != Maximum number of generations per GA run
      BM = 0.08                != Begin mutation probability
      EM = 0.05                != End mutation probability
      FM = 1.0                 != Factor for varying mutation
      nSTIPE=0                 != Selection type option
      CP = 5                   != Exponent(c) for ranking method
      NTOUR=10                 != Number of strings in tournament
      DELTAXDELING=3.0D0       != Variable bounds decreasing factor
      EINDGRENSGROOOTE=1.0D-4 != Maximum variable bound size
      NVSK=5                   != No improvement per run stopping criteria
      STOPHER=1.d-6
C *************************** INPUT PARAMETERS ********************** C
C ********************************************************************** C
C                           MAIN PROGRAM BELOW                           C
C ********************************************************************** C
      DO I=1,N
      GRENS(I,1)=BL(I)
      GRENS(I,2)=BU(I)
```

```
      ENDDO
      NPOPBEG=NPOP
      FMBEG=FM
      EMBEG=EM
      PFBBEG=PFB
      PFEBEG=PFE
      NGAFEVAL=0
C  ************* DETERMINE NUMBER OF INDEPENDANT GA RUNS ************* C
      MAXDX=0.d0
      DO I=1,N
      DELTAX(I)=GRENS(I,2)-GRENS(I,1)
      IF (DELTAX(I).GT.MAXDX)  MAXDX=DELTAX(I)
      ENDDO
      HERHAAL=IDNINT(log(MAXDX/EINDGRENSGROOOTE)/log(DELTAXDELING)+1.D0)
      IF (HERHAAL.LT.0.1D-1) THEN
      HERHAAL=1
      ENDIF
C  ************* DETERMINE NUMBER OF INDEPENDANT GA RUNS ************* C
C
      DO 5000,nHER=1,HERHAAL            ! INDEPENDANT GA RUNS LOOP BEGIN
  111 CONTINUE
      DO I=1,N
      DELTAX(I)=GRENS(I,2)-GRENS(I,1)
      IF (DELTAX(I).LT.EINDGRENSGROOOTE) THEN
      RETURN
      ENDIF
      ENDDO
C ********************* INITIALIZE PARAMETERS ************************* C
      GEBIED=2**NBIN-1
      DNA=N*NBIN
      DELTAM=(EM-BM)/dble(NGEN-1)
      DELTAPF=(PFE-PFB)/dble(NGEN)
      PL=0
      M=0
C ********************* INITIALIZE PARAMETERS ************************* C
C ******** DETERMINE RELATIVE FITNESS IN RANKING SELECTION *********** C
      SOMPERS=0
      DO 10,I=1,NPOP
      PERS(I)=2*dble(NPOP+1-I)**CP/dble(NPOP**2+NPOP)
      SOMPERS=SOMPERS+PERS(I)
   10 CONTINUE
      SOM=0
      DO 15,I=1,NPOP
      SOM=SOM+PERS(I)/SOMPERS
      PERS(I)=SOM
   15 CONTINUE
C ******** DETERMINE RELATIVE FITNESS IN RANKING SELECTION *********** C
C ********************** GENERATE FIRST POPULATION ****************** C
```

```
      DO 18,I=1,NPOP
      DO 17,J=1,DNA
      CALL RANMAR(P,1)
      P=NINT(P)
      POP(I,J)=P
   17 CONTINUE
   18 CONTINUE
C ********************** GENERATE FIRST POPULATION ****************** C
      MINNR=0
C *********** NUMBER OF GENERATIONS PER GA RUN LOOP BEGIN ************ C
   19 M=M+1
C ******* CALCULATE CHANGING PENALTY FACTOR FOR CONSTRAINTS ********** C
      IF (PL.EQ.0) THEN
      PENFAK=PFB+DELTAPF*5*M/6.0d0
      PL=1
      ELSE
      PENFAK=PFB+DELTAPF*M
      PL=0
      ENDIF
C ******* CALCULATE CHANGING PENALTY FACTOR FOR CONSTRAINTS ********** C
      MUT=BM+(M-1)*(DELTAM)
      MAKSMUTC=NBIN-IDNINT(M*(dble(NBIN)-dble(NBIN)/FM)/dble(NGEN))
C ************* EVALUATE POPULATION OF STRINGS BEGIN ***************** C
      DO 30,I=1,NPOP
         VOLGORDE(I)=I
C ******* DETERMINE X-VALUE CORESPONDING TO BINARY REPRESENTATION **** C
         DO 25,II=1,N
            XX(I,II)=0
            DO 20,J=1,NBIN
            XX(I,II)=XX(I,II)+POP(I,NBIN*(II-1)+J)*2**(J-1)
   20       CONTINUE
          XX(I,II)=XX(I,II)*(GRENS(II,2)-GRENS(II,1))/GEBIED+GRENS(II,1)
         X(II)=XX(I,II)
   25    CONTINUE
C ******* DETERMINE X-VALUE CORESPONDING TO BINARY REPRESENTATION **** C
      CALL FUN(N,X,F)                !  FUNCTION EVALUATION FOR VECTOR X
      FC(I)=F
      NGAFEVAL=NGAFEVAL+1
      IF (NI.GT.0) THEN
      CALL CONIN(N,NI,X,C)! INEQUALITY CONSTRAINTS EVALUATION FOR VECTOR X
      DO CI=1,NI
      IF (C(CI).GT.0) THEN
      F=F+PENFAK*C(CI)**2
      ENDIF
      ENDDO
      ENDIF
      IF (NE.GT.0) THEN
      CALL CONEQ(N,NE,X,H)! EQUALITY CONSTRAINTS EVALUATION FOR VECTOR X
```

```
      DO CI=1,NE
      F=F+PENFAK*H(CI)**2
      ENDDO
      ENDIF
      FUNK(I,1)=F
      FUNK(I,2)=I
      FF(I,1)=FUNK(I,1)
      FF(I,2)=I
C ************ UPDATING OF GLOBAL BEST FUNCTION VALUE BEGIN ********** C
      IF (NGAFEVAL.EQ.1) THEN
      FMIN=F
      DO KKL=1,N
      XMIN(KKL)=X(KKL)
      ENDDO
      ELSEIF (F.LT.FMIN) THEN
      FMIN=F
      DO KKL=1,N
      XMIN(KKL)=X(KKL)
      ENDDO
      ENDIF
C ************ UPDATING OF GLOBAL BEST FUNCTION VALUE END ************ C
   30 CONTINUE
C ************** EVALUATE POPULATION OF STRINGS END ****************** C
C ***** ARRANGE FUNCTION VALUES FROM SMALLEST TO LARGEST ************** C
   35 DO 40,RY=2,NPOP
      RYNR=RY
   38 IF (RYNR.EQ.1) THEN
      ELSE IF (FUNK(RYNR,1).LT.FUNK(RYNR-1,1)) THEN
      FUNK(RYNR-1,1)=FUNK(RYNR,1)
      FUNK(RYNR-1,2)=FUNK(RYNR,2)
      FUNK(RYNR,1)=FF(RYNR-1,1)
      FUNK(RYNR,2)=FF(RYNR-1,2)
      FF(RYNR-1,1)=FUNK(RYNR-1,1)
      FF(RYNR-1,2)=FUNK(RYNR-1,2)
      FF(RYNR,1)=FUNK(RYNR,1)
      FF(RYNR,2)=FUNK(RYNR,2)
      RYNR=RYNR-1
      GOTO 38
      ELSE
      END IF
   40 CONTINUE
C ***** ARRANGE FUNCTION VALUES FROM SMALLEST TO LARGEST ************** C
C***************** STOPPING CRITERIA FOR CURRENT GA RUN ***************** C
      F=FC(FUNK(1,2))
      VERSKIL=DABS(F-FMINWAR)
      IF (M.EQ.1) THEN
      FMINWAR=F
      ELSEIF (F.LT.FMINWAR) THEN
```

```
      FMINWAR=F
        IF (VERSKIL.LT.1.D-3) THEN
         MINNR=MINNR+1
        ELSE
        MINNR=1
        ENDIF
      ELSEIF (VERSKIL.LT.1.D-3) THEN
      MINNR=MINNR+1
      ENDIF
      IF (MINNR.GT.NVSK) THEN
      M=NGEN+1
      ENDIF
C*************** STOPPING CRITERIA FOR CURRENT GA RUN *************** C
C
C ********************* SELECTION BEGIN  ***************************** C
C
C ********************** RANKING SELECTION ************************** C
      IF (NSTIPE.EQ.0) THEN
      DO 50,I=1,NPOP
   42 CALL RANMAR(P,1)
      RY=1
   45 IF (P.LT.PERS(RY)) THEN
      DO 47,J=1,DNA
      POPNR=FUNK(RY,2)
      PPOP(I,J)=POP(POPNR,J)
   47 CONTINUE
      ELSE
      RY=RY+1
      GOTO 45
      END IF
   50 CONTINUE
C ********************** RANKING SELECTION ************************** C
C
C  ******************** TOURNAMENT SELECTION ************************** C
      ELSEIF (NSTIPE.EQ.1) THEN
      DO I=1,NPOP
      DO J=1,NTOUR
      CALL RANMAR(P,1)
      P=NINT(NPOP*P)
      IF (P.EQ.0) THEN
      P=NPOP
      ENDIF
      TOURF(J,1)=FUNK(P,2)
      TOURF(J,2)=FUNK(P,1)
      ENDDO
      RY=NTOUR
      MINTSNR=TOURF(RY,1)
      MINTSV=TOURF(RY,2)
```

```
    9 IF (RY.EQ.1) THEN
      ELSEIF (MINTSV.LT.TOURF(RY-1,2)) THEN
      RY=RY-1
      GOTO 9
      ELSE
      MINTSNR=TOURF(RY-1,1)
      MINTSV=TOURF(RY-1,2)
      RY=RY-1
      GOTO 9
      ENDIF
      DO J=1,DNA
      PPOP(I,J)=POP(MINTSNR,J)
      ENDDO
      ENDDO
      ENDIF
C  ******************** TOURNAMENT SELECTION ************************* C
C
C ********************** SELECTION END  ******************************** C
      DO 52,J=1,NPOP
      DO 51,I=1,DNA
      POP(J,I)=PPOP(J,I)
   51 CONTINUE
   52 CONTINUE
C ********************** CROSSOVER BEGIN ****************************** C
      DO 60,I=1,NPOP/2
C ********************** CHOOSE FIRST MATING PARENT RANDOMLY ********** C
      CALL RANMAR(P,1)
      P=NINT((NPOP-(I-1)*2)*P)
      IF (P.EQ.0) THEN
      P=(NPOP-(I-1)*2)
      END IF
      MNR(1)=VOLGORDE(P)
      K=0
      DO 55,J=1,(NPOP-(I-1)*2)
      IF (J.NE.P) THEN
      VOLGORDE(J-K)=VOLGORDE(J)
      ELSE
      K=1
      END IF
   55 CONTINUE
C ********************** CHOOSE FIRST MATING PARENT RANDOMLY ********** C
C ********************** CHOOSE SECOND MATING PARENT RANDOMLY ********** C
      CALL RANMAR(P,1)
      P=NINT((NPOP-(I-1)*2-1)*P)
      IF (P.EQ.0) THEN
      P=(NPOP-(I-1)*2-1)
      END IF
      MNR(2)=VOLGORDE(P)
```

```
      K=0
      DO 56,J=1,(NPOP-(I-1)*2-1)
      IF (J.NE.P) THEN
      VOLGORDE(J-K)=VOLGORDE(J)
      ELSE
      K=1
      END IF
   56 CONTINUE
C ******************** CHOOSE SECOND MATING PARENT RANDOMLY ********** C
      CALL RANMAR(P,1)
      P=NINT((DNA-1)*P)
      IF (P.EQ.0) THEN
      P=DNA-1
      END IF
      CROSPOS=P                              ! CROSSOVER POSITION SELECTED
      DO 57,J=CROSPOS+1,DNA
      POP(MNR(1),J)=PPOP(MNR(2),J)
      POP(MNR(2),J)=PPOP(MNR(1),J)
   57 CONTINUE
C ********************* CROSSOVER END ******************************** C
C  ************************* MUTATION ******************************** C
      DO 59,J=1,2
      RYMUT=MNR(J)
      DO KI=1,N
      DO 58,II=1,MAKSMUTC
      CALL RANMAR(MUTW,1)
      IF (MUTW.LT.MUT) THEN
      KOLMUT=(KI-1)*NBIN+II
      IF (POP(RYMUT,KOLMUT).EQ.1) THEN
      POP(RYMUT,KOLMUT)=0
      ELSE
      POP(RYMUT,KOLMUT)=1
      END IF
      END IF
   58 CONTINUE
      ENDDO
   59 CONTINUE
C  ************************* MUTATION ******************************** C
   60 CONTINUE
 1000 IF (M.LT.NGEN) THEN
      GOTO 19
      ENDIF
C *********** NUMBER OF GENERATIONS PER GA RUN LOOP BEGIN ************ C
C ***************** TEST FOR CONSTRAINT VIOLATION ****************** C
      DO I=1,N
      X(I)=XX(FUNK(1,2),I)
      ENDDO
      CALL CONIN(N,NI,X,C)
```

```
      CALL CONEQ(N,NI,X,H)
      IVERANDERPF=0
      DO KL=1,NI
      IF (C(KL).GT.2.D0) IVERANDERPF=1
      ENDDO
      DO KL=1,NE
      IF (H(KL).GT.2.D0) IVERANDERPF=1
      ENDDO
      IF (IVERANDERPF.EQ.1) THEN
      PFB=PFB+20.D0
      PFE=PFE+20.D0
      GOTO 111
      ENDIF
C ****************** TEST FOR CONSTRAINT VIOLATION ****************** C
C ****************DETERMINE NEW BOUNDS FOR FOLLOWING GA RUN ********** C
      DO I=1,N
      GRENS(I,1)=X(I)-DELTAX(I)/(DELTAXDELING*2.d0)
      GRENS(I,2)=X(I)+DELTAX(I)/(DELTAXDELING*2.d0)
      ENDDO
C  *********** ENSURE THAT NEW BOUNDS STAY WITHIN PROBLEM BOUNDS ***** C
      DO I=1,N
      IF (GRENS(I,1).LT.BL(I)) THEN
      GRENS(I,1)=BL(I)
      IF (GRENS(I,1).GT.GRENS(I,2)) GRENS(I,2)=GRENS(I,1)+0.1d-2
      ENDIF
      IF (GRENS(I,2).GT.BU(I)) THEN
      GRENS(I,2)=BU(I)
      IF (GRENS(I,1).GT.GRENS(I,2)) GRENS(I,1)=GRENS(I,2)-0.1d-2
      ENDIF
      ENDDO
C  *********** ENSURE THAT NEW BOUNDS STAY WITHIN PROBLEM BOUNDS ***** C
C ****************DETERMINE NEW BOUNDS FOR FOLLOWING GA RUN ********** C
C ************************** TEST EARLY STOPPING CONDITION *********** C
      IF (NHER.NE.1.AND.dabs(FBEST-FMINWAR).LT.STOPHER) GOTO 1111
      FBEST=FMINWAR
C ************************** TEST EARLY STOPPING CONDITION *********** C
 5000 CONTINUE
C ******************** END OF NUMBER OF GA RUNS LOOP ***************** C
 1111 DO I=1,N
      X(I)=XX(FUNK(1,2),I)
      ENDDO
      NPOP=NPOPBEG
      FM=FMBEG
      EM=EMBEG
      PFB=PFBBEG
      PFE=PFEBEG
      F=FMIN
      DO KLL=1,N
```

```
      X(KLL)=XMIN(KLL)
      ENDDO
      RETURN
      END
C ******************************** END ******************************** C
C ********************************************************************** C
```

F.2 Particle Swarm Optimization Algorithm

```
C ********************************************************************** C
C ********************************************************************** C
C                                                                        C
C             PARTICLE SWARM OPTIMIZATION BY H.P.J.BOLTON                C
C                                                                        C
C ********************************************************************** C
C ********************************************************************** C
      SUBROUTINE PSO(X,N,F,BL,BU)
      IMPLICIT REAL*8(A-H,O-Z),INTEGER(I-N)
      INCLUDE 'params.inc'
      DIMENSION X(N),BL(N),BU(N)
      DOUBLE PRECISION XPSO(MAXPOP,MAXSIZ),XBEST(MAXSIZ)
      DOUBLE PRECISION XPSOBEST(MAXPOP,MAXSIZ),FPSOBEST(MAXPOP)
      DOUBLE PRECISION DELTAXPSO(MAXPOP,MAXSIZ),FPSO(MAXPOP)
C
C *************************** INPUT PARAMETERS ********************** C
      NPSO=15
      C1=1.d0
      C2=1.d0
      WBEG=0.8d0
      STEPMAX=30.D0
      NSLOOPSMAX=4000
      NHEHAALMAX=20
C *************************** INPUT PARAMETERS ********************** C
C ***************** INITIALIZE OF BEGINNING VECTORS BEGIN ************* C
      DO I=1,NPSO
        DO JI=1,N
          CALL RANMAR(P,1)
          XPSO(I,JI)=BL(JI)+P*(BU(JI)-BL(JI))
          DELTAXPSO(I,JI)=0.D0
        ENDDO
      ENDDO
C ***************** INITIALIZE OF BEGINNING VECTORS END *************** C
      NPSLOOPS=0
  50  NPSLOOPS=NPSLOOPS+1
      W=WBEG
C
      DO I=1,NPSO
         DO II=1,N
          X(II)=XPSO(I,II)
         ENDDO
         CALL FUN(N,X,F)             !  FUNCTION EVALUATION FOR VECTOR X
         FPSO(I)=F
         NPSKOUNTS=NPSKOUNTS+1
C ************ UPDATING OF GLOBAL BEST FUNCTION VALUE BEGIN ********** C
```

```
          IF (NPSLOOPS.EQ.1.AND.I.EQ.1) THEN
           FBEST=F
           DO KI=1,N
             XBEST(KI)=X(KI)
           ENDDO
          ELSEIF (F.LT.FBEST) THEN
           FBEST=F
           DO KI=1,N
             XBEST(KI)=X(KI)
           ENDDO
          ENDIF
C ************ UPDATING OF GLOBAL BEST FUNCTION VALUE END ************ C
C ************ UPDATING OF PARTICLE BEST FUNCTION VALUE BEGIN ******** C
          IF (NPSLOOPS.EQ.1) THEN
           FPSOBEST(I)=F
           DO KI=1,N
             XPSOBEST(I,KI)=X(KI)
           ENDDO
          ELSEIF (F.LT.FPSOBEST(I)) THEN
           FPSOBEST(I)=F
           DO KI=1,N
             XPSOBEST(I,KI)=X(KI)
           ENDDO
          ENDIF
C ************ UPDATING OF PARTICLE BEST FUNCTION VALUE END ********** C
      ENDDO
C
C *************** DETERMINE NEW PARTICLE POSITIONS BEGIN ************* C
      DO I=1,NPSO
         CALL RANMAR(R1,1)
         CALL RANMAR(R2,1)
         DO II=1,N
          DELTAXPSO(I,II)=DELTAXPSO(I,II)*W+R1*C1*(XBEST(II)-XPSO(I,II))
     #    +R2*C2*(XPSOBEST(I,II)-XPSO(I,II))
          IF (DELTAXPSO(I,II).GT.STEPMAX) DELTAXPSO(I,II)=STEPMAX
          IF (DELTAXPSO(I,II).LT.-STEPMAX) DELTAXPSO(I,II)=-STEPMAX
          XPSO(I,II)=XPSO(I,II)+DELTAXPSO(I,II)
C ******************* KEEP THE VARIABLE WITHIN BOUNDS *************** C
          IF (XPSO(I,II).LT.BL(II)) THEN
           XPSO(I,II)=BL(II)
          ENDIF
          IF (XPSO(I,II).GT.BU(II)) THEN
           XPSO(I,II)=BU(II)
          ENDIF
C ******************* KEEP THE VARIABLE WITHIN BOUNDS *************** C
         ENDDO
      ENDDO
C *************** DETERMINE NEW PARTICLE POSITIONS BEGIN ************* C
```

```
      IF (NPSLOOPS.EQ.1) THEN
      NSAMEBEST=1
      FSAMEBEST=FBEST
      ELSEIF (DABS(FSAMEBEST-FBEST).LT.1.D-5) THEN
      NSAMEBEST=NSAMEBEST+1
        IF (FBEST.LT.FSAMEBEST) THEN
        FSAMEBEST=FBEST
        ENDIF
      ELSEIF (FBEST.LT.FSAMEBEST) THEN
      NSAMEBEST=1
      FSAMEBEST=FBEST
      ENDIF
      IF (NPSLOOPS.EQ.NSLOOPSMAX.OR.NSAMEBEST.EQ.NHEHAALMAX) THEN
      F=FSAMEBEST
         DO KI=1,N
           X(KI)=XBEST(KI)
         ENDDO
      RETURN
      ELSE
      GOTO 50
      ENDIF
      END
C ******************************** END ******************************** C
C ********************************************************************** C
```

F.3 Master program for parallel optimization infrastructure

```
C ********************************************************************** C
C MULTI OPTIMIZATION METHOD FOR UNCONSTAINED AND CONSTRAINED PROBLEMS  C
C                                                                      C
C                 PROGRAMMED BY H.P.J.BOLTON                           C
C                                                                      C
C                   UNIVERSITY OF PRETORIA                             C
C                                                                      C
C             DEPARTMENT OF MECHANICAL ENGINEERING                     C
C                                                                      C
C                  THIS VERSION:  22 AUGUST 2000                       C
C                                                                      C
C ********************************************************************** C
      PROGRAM MASTER
      IMPLICIT REAL*8(A-H,O-Z),INTEGER(I-N)
      INCLUDE 'params.inc'
C ******************* STANDARD PARALLEL DECLARATIONS ***************** C
      include '../include/fpvm3.h'
      integer i, info, nproc, nhost, msgtype
      integer mytid, iptid, dtid, tids(128),me
      character*18 nodename, host
      character*8 arch
C ******************* STANDARD PARALLEL DECLARATIONS ***************** C
      DOUBLE PRECISION X(MAXSIZ),BL(MAXSIZ),BU(MAXSIZ)
      DOUBLE PRECISION GF(MAXSIZ),XMIN(MAXSIZ)
      DOUBLE PRECISION C(MAXCON),H(MAXCON)
      double precision probil(15)
      integer nmaxfunct(128),nmaxgradt(128)
      double precision  contrn(15),contrr(15)
      double precision  tcontrn(15),tcontrr(15)
      character*1 converged*1
      dimension iprobmap(56)
      COMMON /PRIORI/ FAPRIORI
      COMMON /PRDATA/ NPROB
      dimension nalgorithm(15)
      data nalgorithm  /1,3,5,6,10,14,15,16,17,19,20,21,25,26,27/
      data iprobmap /1,2,4,6,8,10,13,22,23,24,25,26,
     &               47,48,49,
     &               27,28,38,39,
     &               29,30,31,32,33,
     &               34,35,36,37,
     &               44,45,46,
     &               0,0,0,
     &               0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
     &               50,51,52,53,54,55,56/
```

```
C
C********************** RECEIVE PROBLEM INFORMATION FROM DATAFILE **** C
      NAMELIST / INLIG   / METHOD,MULTIJANEE,MAXITER,MINITER
     &,PROBSTOP,NREPEATS,NBPROB,NEPROB,FSELLE,CONVERG,nproc
      OPEN (UNIT=23, FILE='datafile.inp', STATUS='OLD')
      REWIND(23)
      READ (23, NML = INLIG)
      CLOSE(23)
C********************** RECEIVE PROBLEM INFORMATION FROM DATAFILE **** C
      call timer (time01)
C *****************  INITIALIZE THE MPPVM WITH SLAVES *************** C
      call pvmfmytid( mytid )  ! give master an id (mtid=mytid)
      call pvmfconfig( nhost, narch, dtid, host, arch, speed, info )
      nhost=nproc
      write(*,*) nhost,' hosts detected in configuration.'
C *****************  INITIALIZE THE MPPVM WITH SLAVES *************** C
C ***************** INITIALIZE THE RANDOM NUMBER GENERATOR *********** C
      OPEN(UNIT=20,FILE='random.def')
      READ(20,*) ISEED
      READ(20,*) JSEED
      CLOSE(20)
      iseed = 1802
      jseed = 9373
      CALL RMARIN(ISEED,JSEED)
      ISEED=ISEED+7
      IF (ISEED.GT.31328) THEN
      ISEED=ISEED-31328
      ENDIF
      JSEED=JSEED+3
      IF (JSEED.GT.30081) THEN
      JSEED=JSEED-30081
      ENDIF
      OPEN(UNIT=20,FILE='random.def')
      WRITE(20,*) ISEED
      WRITE(20,*) JSEED
      CLOSE(20)
C ***************** INITIALIZE THE RANDOM NUMBER GENERATOR *********** C
C ***************** INITIALIZE THE OUTPUT FILES ********************** C
      OPEN(1,FILE='results.out')
      WRITE(6,3000)
      WRITE(1,3000)
      WRITE(1,2000)
      open (65,file='averages.out',status='unknown')
      open (70,file='variables.out',status='unknown')
      OPEN(UNIT=45,FILE='algprob.out')
      OPEN(UNIT=55,FILE='contribute.out')
      write(55,1201) (nalgorithm(ki),ki=1,15)
      write(45,1202) (nalgorithm(ki),ki=1,15)
```

```
C ***************** INITIALIZE THE OUTPUT FILES ********************** C
C
      NRPROBS=0       ! TOTAL NUMBER OF PROBLEMS DONE
      NFAIL=0         ! TOTAL NUMBER OF FAILURES TO CONVERGE
      NTNFEVALS=0     ! TOTAL NUMBER OF NORMILIZED FUNCTION EVALUATIONS
      NTFEVAL=0       ! TOTAL OF PURE FUNCTION EVALUATIONS
      NTGRADFEVALS=0 ! TOTAL OF FUNC EVALS DUE TO GRAD VECT EVALS
C
      DO 140,NRI=NBPROB,NEPROB
C
      NAVEFEVALS=0     ! AVERAGE NUMBER OF FUNCTION EVALS
      NAVECOMP=0       ! AVERAGE NUMBER OF COMPOUND EVALS
      FMINAVE=0        ! AVERAGE FUNCTION VALUE
      NAVER=0          ! NUMBER OF TIMES R
      NAVEN=0          ! NUMBER OF TIMES N
      do ki=1,15
      tcontrn(ki)=0.d0
      tcontrr(ki)=0.d0
      enddo
      NCONVERG=0      ! AVERAGE NUMBER OF CONVERGENGE
      NPROB=iprobmap(NRI)        ! PROBLEM NUMBER BEING OPTIMIZED
C
      CALL  GETPROBNO (N,BL,BU,NI,NE) ! VERKRY PROBLEEM INLIGTING
C ******************** PROBLEM REPEATS LOOP BEGIN ******************** C
      DO 138,JJ=1,NREPEATS
      call timer (time1)
      do kn=1,128
       nmaxfunct(kn)=0
       nmaxgradt(kn)=0
      enddo
      do ki=1,15
      contrn(ki)=0.d0
      contrr(ki)=0.d0
      enddo
      NGRADEVAL=0
      NFUNCEVAL=0
      NRPROBS=NRPROBS+1
      ndoprob=0
      write(45,*) 'Prob. = ',nprob,'  Nr = ',jj
C ******************** STARTING PROBABILITIES FOR ALGORITHMS ********* C
      if (ni.eq.0.and.ne.eq.0) then
       probil(1)=0.15D0    ! PROB THAT MBB-BFGS              IS CHOSEN
       probil(2)=0.15D0    ! PROB THAT BFGS                  IS CHOSEN
       probil(3)=0.05D0    ! PROB THAT SQSD99                IS CHOSEN
       probil(4)=0.0D0     ! PROB THAT ETOPC99               IS CHOSEN
       probil(5)=0.00D0    ! PROB THAT CARROL-GA             IS CHOSEN
       probil(6)=0.07D0    ! PROB THAT BOLTON-GA             IS CHOSEN
       probil(7)=0.00D0    ! PROB THAT CONSTRAINED MBB-BFGS  IS CHOSEN
```

```
       probil(8)=0.05D0    ! PROB THAT CONSTRAINED LEAPFROG  IS CHOSEN
       probil(9)=0.05D0    ! PROB THAT LEAPFROG              IS CHOSEN
       probil(10)=0.07D0   ! PROB THAT MOCKUS                IS CHOSEN
       probil(11)=0.06D0   ! PROB THAT PARTICLE SWORM        IS CHOSEN
       probil(12)=0.10D0   ! PROB THAT CLUSTERING            IS CHOSEN
       probil(13)=0.1D0    ! PROB THAT POLACK RIBIER         IS CHOSEN
       probil(14)=0.05D0   ! PROB THAT DYNAMIC-Q             IS CHOSEN
       probil(15)=0.1D0    ! PROB THAT MBB-POLACK RIBIER     IS CHOSEN
      else
       probil(1)=0.0D0     ! PROB THAT MBB-BFGS              IS CHOSEN
       probil(2)=0.0D0     ! PROB THAT BFGS                  IS CHOSEN
       probil(3)=0.0D0     ! PROB THAT SQSD99                IS CHOSEN
       probil(4)=0.25D0    ! PROB THAT ETOPC99               IS CHOSEN
       probil(5)=0.0D0     ! PROB THAT CARROL-GA             IS CHOSEN
       probil(6)=0.0D0     ! PROB THAT BOLTON-GA             IS CHOSEN
       probil(7)=0.25D0    ! PROB THAT CONSTRAINED MBB-BFGS  IS CHOSEN
       probil(8)=0.25D0    ! PROB THAT CONSTRAINED LEAPFROG  IS CHOSEN
       probil(9)=0.0D0     ! PROB THAT LEAPFROG              IS CHOSEN
       probil(10)=0.0D0    ! PROB THAT MOCKUS                IS CHOSEN
       probil(11)=0.0D0    ! PROB THAT PARTICLE SWORM        IS CHOSEN
       probil(12)=0.0D0    ! PROB THAT CLUSTERING            IS CHOSEN
       probil(13)=0.0D0    ! PROB THAT POLACK RIBIER         IS CHOSEN
       probil(14)=0.25D0   ! PROB THAT DYNAMIC-Q             IS CHOSEN
       probil(15)=0.0D0    ! PROB THAT MBB-POLACK RIBIER     IS CHOSEN
      endif
C ******************** STARTING PROBABILITIES FOR ALGORITHMS ********* C
C
C ************** SNYMAN-FATTI STOPPING CONDITION LOOP BEGIN ********** C
      J=0
      NR=1
   20 continue
      nodename = 'slave'
      arch = '*'                              ! '*' random slave calling
      call pvmfspawn( nodename, PVMDEFAULT, arch, nproc, tids, numt )
      nmaxtids=tids(nproc)
C
C ********************* CHECK FOR SPAWNING PROBLEMS ****************** C
      if( numt .lt. nproc ) then
         print *, 'trouble spawning ',nodename
         print *, ' Check tids for error code'
         call shutdown( numt, tids )
      endif
C ********************* CHECK FOR SPAWNING PROBLEMS ****************** C
C
C ********************* SEND DATA FOR EACH SLAVE ********************* C
      do indexslave = 1, nproc
C
      call GETRANDOM (N,X,bl,bu)            ! GET RANDOM STARTING VECTOR
```

```
C
      if(MULTIJANEE.eq.1) then
      call CHOOSEMETHOD(METHOD,probil)   ! DETERMINE OPT. ALGORITHM USED
      endif
C
      call pvmfinitsend( PVMDEFAULT, info )
      call pvmfpack( INTEGER4,nproc      ,     1, 1, info )
      call pvmfpack( INTEGER4,tids       , nproc, 1, info )
      call pvmfpack( INTEGER4,iseed      ,     1, 1, info )
      call pvmfpack( INTEGER4,jseed      ,     1, 1, info )
      call pvmfpack( INTEGER4,n          ,     1, 1, info )
      call pvmfpack( INTEGER4,nprob      ,     1, 1, info )
      call pvmfpack( INTEGER4,ni         ,     1, 1, info )
      call pvmfpack( INTEGER4,ne         ,     1, 1, info )
      call pvmfpack( INTEGER4,j          ,     1, 1, info )
      call pvmfpack( REAL8,fmin          ,     1, 1, info )
      call pvmfpack( REAL8,bl            ,    20, 1, info )
      call pvmfpack( REAL8,bu            ,    20, 1, info )
      call pvmfpack( REAL8,xmin          ,    20, 1, info )
      call pvmfpack( REAL8,ndoprob       ,     1, 1, info )
      call pvmfpack( INTEGER4,jj         ,     1, 1, info )
      call pvmfpack( REAL8, x            ,    20, 1, info )
      call pvmfpack( INTEGER4,METHOD     ,     1, 1, info )
C
       msgtype = 1
       itoid = tids(indexslave)
      call pvmfsend(itoid,msgtype,info)
      enddo
C ********************* SEND DATA FOR EACH SLAVE ********************* C
C
C ********* RECEIVING OF OPTIMIZATION RESULTS FROM SLAVES BEGIN ****** C
C
      do 30 i=1,maxiter
      khi=0
291   continue
      khi=1+khi
C ********************* RECEIVE RESULTS FROM SLAVE no. = me ********** C
         msgtype  = 2
         call pvmfrecv( -1, msgtype, info )
C
      call pvmfunpack( INTEGER4,me               , 1, 1, info )
      call pvmfunpack( INTEGER4,ntids            , 1, 1, info )
      call pvmfunpack( INTEGER4,NtGRADEVAL       , 1, 1, info )
      call pvmfunpack( INTEGER4,NtFUNCEVAL       , 1, 1, info )
      call pvmfunpack( REAL8, x                  ,20, 1, info )
      call pvmfunpack( REAL8, f                  , 1, 1, info )
      call pvmfunpack( REAL8, dummy              , 1, 1, info )
      call pvmfunpack( INTEGER4,method           , 1, 1, info )
```

```
C ******************** RECEIVE RESULTS FROM SLAVE no. = me ********** C
       if (ntids.lt.tids(1)) goto 291           ! discard pipeline data
      CALL FUN(N,X,F)
      j=j+1
      nmaxfunct(me)=nmaxfunct(me)+NtFUNCEVAL   ! INDIVIDUAL EVALUATIONS
      nmaxgradt(me)=nmaxgradt(me)+NtGRADEVAL   ! INDIVIDUAL EVALUATIONS
C
      write(45,367) (probil(kl),kl=1,15)
C
C ************** CHECK IF SOLUTION FALLS WITHIN BOUNDS *************** C
      DO KI=1,N
      IF ((X(KI)-BU(KI)).GT.1.d-6) f=1.d99
      IF ((BL(KI)-X(KI)).GT.1.d-6) f=1.d99
      ENDDO
C ************** CHECK IF SOLUTION FALLS WITHIN BOUNDS *************** C
C ************** CHECK IF SOLUTION DOES NOT VIOLATE CONSTRAINTS ****** C
        if (ne.gt.0) then
          call coneq(n,ne,x,h)
              do ik=1,ne
               if (dabs(h(Ik)).gt.1.d-6)   f=1.d99
              enddo
        endif
        if (ni.gt.0) then
          call conin(n,ni,x,c)
              do ik=1,ni
               if (c(Ik).gt.1.d-6)   f=1.d99
              enddo
        endif
C ************** CHECK IF SOLUTION DOES NOT VIOLATE CONSTRAINTS ****** C
c
C ************* TEST FOR GLOBAL MINIMUM FOUND SOFAR ******
      kl=0
  71  kl=kl+1
      if (method.ne.nalgorithm(kl)) goto 71
      contrn(kl)=contrn(kl)+1.d0
      F=F-DUMMY
      FVERSKIL=DABS(F-FMIN)
      IF (J.EQ.1) THEN
        FMIN=F
        DO JJK=1,N
        XMIN(JJK)=X(JJK)
        ENDDO
        NR=1
        CALL VNORM (gfnorm,gf,N)         ! CALCULATE GRADIENT VECTOR NORM
        contrr(kl)=contrr(kl)+1.d0
      ELSEIF (F.LT.FMIN) THEN
        IF (FVERSKIL.LT.FSELLE) THEN
          NR=NR+1
```

```
          contrr(kl)=contrr(kl)+1.d0
        ELSE
          NR=1
          do ki=1,15
          contrr(ki)=0.d0
          enddo
          contrr(kl)=contrr(kl)+1.d0
        ENDIF
        FMIN=F
        DO JJK=1,N
        XMIN(JJK)=X(JJK)
        ENDDO
        CALL VNORM (gfnorm,gf,N)        ! CALCULATE GRADIENT VECTOR NORM
      ELSEIF (FVERSKIL.LT.FSELLE) THEN
      NR=NR+1
      contrr(kl)=contrr(kl)+1.d0
      ENDIF
C *************  TEST FOR GLOBAL MINIMUM FOUND SOFAR ******

C
       QNR=CONVPROB(J,NR)      ! CALCULATE FATTI CONVERGENCE PROBABILITY
C
       write(70,5001)  nprob,jj,j,me,method,f,NtFUNCEVAL,NtGRADEVAL,qnr
     &,(x(i),i=1,n)
c
       IF (QNR.lT.PROBSTOP.OR.J.LT.MINITER) then
c
        call GETRANDOM (N,X,bl,bu) ! GET RANDOM STARTING VECTOR
        if(MULTIJANEE.eq.1) then
        call CHOOSEMETHOD(METHOD,probil) ! DETERMINE OPT. ALGORITHM USED
        endif
C ** SEND DATA TO SPECIFIC SLAVE AFTER RECEIVING RESULTS FROM SLAVE ** C
      call pvmfinitsend( PVMDEFAULT, info )
      call pvmfpack( INTEGER4,nproc       ,     1, 1, info )
      call pvmfpack( INTEGER4,tids        , nproc, 1, info )
      call pvmfpack( INTEGER4,iseed       ,  1, 1, info )
      call pvmfpack( INTEGER4,jseed       ,  1, 1, info )
      call pvmfpack( INTEGER4,n           ,  1, 1, info )
      call pvmfpack( INTEGER4,nprob       ,  1, 1, info )
      call pvmfpack( INTEGER4,ni          ,  1, 1, info )
      call pvmfpack( INTEGER4,ne          ,  1, 1, info )
      call pvmfpack( INTEGER4,j           ,  1, 1, info )
      call pvmfpack( REAL8,fmin           ,  1, 1, info )
      call pvmfpack( REAL8,bl             , 20, 1, info )
      call pvmfpack( REAL8,bu             , 20, 1, info )
      call pvmfpack( REAL8,xmin           , 20, 1, info )
      call pvmfpack( REAL8,ndoprob        ,  1, 1, info )
      call pvmfpack( INTEGER4,jj          ,  1, 1, info )
```

```
      call pvmfpack( REAL8, x             , 20, 1, info )
      call pvmfpack( INTEGER4,METHOD      ,  1, 1, info )
C
       msgtype = 1
      itoid=tids(me)
      call pvmfsend(itoid,msgtype,info)
C ** SEND DATA TO SPECIFIC SLAVE AFTER RECEIVING RESULTS FROM SLAVE ** C
      else
         do kill=1, nproc
          call pvmfkill(tids(kill),info)   ! KILL SLAVES AFTER CONVERGED
         enddo
        goto 312
      endif
  30  continue
C ********* RECEIVING OF OPTIMIZATION RESULTS FROM SLAVES END ******** C
C ************** SNYMAN-FATTI STOPPING CONDITION LOOP END ************ C
  312 continue
C ****************** TEST FOR CONVERGENCE TO KNOWN SOLUTION ********** C
      IF (DABS(FAPRIORI).LT.FSELLE) THEN
      FPRIORIVERSKIL=DABS(FMIN-FAPRIORI)
      ELSE
      FPRIORIVERSKIL=DABS((FMIN-FAPRIORI)/FAPRIORI)
      ENDIF
      IF (FPRIORIVERSKIL.GT.CONVERG) THEN
      NFAIL=NFAIL+1
      CONVERGED='N'
      ELSE
      CONVERGED='C'
      ENDIF
      IF (FMIN.LT.(FAPRIORI-CONVERG)) THEN
      CONVERGED='L'
      ENDIF
C ****************** TEST FOR CONVERGENCE TO KNOWN SOLUTION ********** C
      IF (CONVERGED.EQ.'C'.OR.CONVERGED.EQ.'L') THEN
      NCONVERG=NCONVERG+1
      ENDIF
      ntotalmaxfunk=0
      ntotalmaxgrad=0
      ntotalmaxnorm=0                              ! normalized cost
      do kn=1,128
      nnorme=nmaxfunct(kn)+n*nmaxgradt(kn)        ! normalized cost
      if (nnorme.gt.ntotalmaxnorm) then           ! normalized cost
      ntotalmaxnorm=nnorme                        ! normalized cost
c      if (nmaxfunct(kn).gt.ntotalmaxfunk) then   ! function evaluations
      ntotalmaxfunk=nmaxfunct(kn)
      ntotalmaxgrad=nmaxgradt(kn)
      endif
      enddo
```

```
      NFUNCEVAL=ntotalmaxfunk
      NGRADEVAL=ntotalmaxgrad
C ************* UPDATE NUMBER OF FUNCTION EVALUATION TOTALS ********** C
      NTFEVAL=NTFEVAL+NFUNCEVAL
      NFEVAL=NFUNCEVAL+(N)*NGRADEVAL
      NTNFEVALS=NTNFEVALS+NFEVAL
      NTGRADFEVALS=NTGRADFEVALS+(N)*NGRADEVAL
      NAVEFEVALS=NAVEFEVALS+NFUNCEVAL                  !  FUNCTION COST
      NAVECOMP=NAVECOMP+NFUNCEVAL+(N)*NGRADEVAL        !  COMPOUND COST
      NAVER=NAVER+NR
      NAVEN=NAVEN+J
      FMINAVE=FMINAVE+FMIN
      IF (JJ.EQ.1) THEN
      FBEST=FMIN
      ELSEIF (FMIN.LT.FBEST) THEN
      FBEST=FMIN
      ENDIF
C ************* UPDATE NUMBER OF FUNCTION EVALUATION TOTALS ********** C
c ************************* PRINTING RESULTS ************************** C
      call timer(time2)
      elapsed=(time2-time1)
      fdifference=DABS(FMIN-FAPRIORI)
      write(*,2001) nprob,fmin,NFUNCEVAL,NGRADEVAL,NFEVAL,NR,J,
     &              converged,elapsed
      IF (N.LE.11)
     & write(1,1000) nprob,n,converged,fmin,fdifference,gfnorm,NFUNCEVAL
     &,NGRADEVAL,nr,j,NCONVERG,qnr,(xmin(i),i=1,N)
      IF (N.GT.11)
     & write(1,1001) nprob,n,converged,fmin,fdifference,gfnorm,NFUNCEVAL
     &,NGRADEVAL,nr,j,NCONVERG,qnr,(xmin(i),i=1,10)
c ************************* PRINTING RESULTS ************************** C
      do ki=1,15
      tcontrn(ki)=tcontrn(ki)+contrn(ki)
      tcontrr(ki)=tcontrr(ki)+contrr(ki)
      enddo
  138 ENDDO
C ******************** PROBLEM REPEATS LOOP END ********************** C
       NAVER=idnint( dble(NAVER)/dble(NREPEATS))
       NAVEN=idnint( dble(NAVEN)/dble(NREPEATS))
       Qdum=CONVPROB(NAVEN,NAVER)    ! CALCULATE CONVERGENCE PROBABILITY
        WRITE(65,665) nprob,NAVEFEVALS/NREPEATS,NAVECOMP/NREPEATS
     &,NREPEATS-NCONVERG,NAVER,NAVEN,Qdum
     &,dabs(FMINAVE/DBLE(NREPEATS)-FAPRIORI),dabs(FBEST-FAPRIORI)
      do ki=1,15
      tcontrn(ki)=tcontrn(ki)/dble(NREPEATS)
      tcontrr(ki)=tcontrr(ki)/dble(NREPEATS)
       enddo
      write(55,3467) nprob,(tcontrr(ki),tcontrn(ki),ki=1,15)
```

```
  140 ENDDO
C ******************* SUMMARY OF ANALYSIS RESULTS ******************** C
      call timer (time02)
      elapsed=(time02-time01)
      write(6,2002) NTFEVAL,NTFEVAL/NRPROBS,nfail,elapsed,NTGRADFEVALS,
     &                NTNFEVALS/NRPROBS
      write(1,2002) NTFEVAL,NTFEVAL/NRPROBS,nfail,elapsed,NTGRADFEVALS,
     &                NTNFEVALS/NRPROBS
      epsmch1=dpmeps()
      epsmch=1.d-15
      write(6,*)    '  Machine precision = ',epsmch1
      write(1,*)  '  Machine precision = ',epsmch1
      write(6,*)    '  Current precision = ',epsmch
      write(1,*)  '  Current precision = ',epsmch
      write(6,*)    ' '
      write(1,*)  ' '
C ******************  SUMMARY OF ANALYSIS RESULTS ****************** C
      CLOSE(65)
      CLOSE(1)
      close(70)
      CLOSE(45)
      CLOSE(55)
       call pvmfexit(info)  ! leave PVM before exiting
      STOP
C ********************** FORMATS FOR WRITE *************************** C
 367  format(1f5.2,'  ',1f5.2,'  ',1f5.2,'  ',1f5.2,'  ',1f5.2,'  '
     &,1f5.2,'  ',1f5.2,'  ',1f5.2,'  ',1f5.2,'  ',1f5.2,'  ',1f5.2,'  '
     &,1f5.2,'  ',1f5.2,'  ',1f5.2,'  ',1f5.2,'  ',1f5.2,'  ',1f5.2,'  '
     &)
 665   format (' AVE. PROBLEM = ',i4,' FUNC = ',i9,
     &' COMPOUND = ',i9,'  #FAIL = ',i4,' r =',i2,' n =',i3,
     &' qnr=',f7.4,' FMINAVE=',e12.5,' FBEST=',e12.5)
 1000 format (i3,i4,4x,a1,1x,3e14.5,2i6,1i3,' /',2i4,f7.4,11e14.5)
 1001 format (i3,i4,4x,a1,1x,3e14.5,2i6,1i3,' /',2i4,f7.4,10e14.5,
     &        ' ... etc. ...')
 1201 format('            ','Alg.',I2,'     Alg.',I2,'     Alg.',I2
     &,'     Alg.',I2,'     Alg.',I2,'     Alg.',I2,'     Alg.',I2
     &,'     Alg.',I2,'     Alg.',I2,'     Alg.',I2,'     Alg.',I2)
 1202 format('Alg.',I2,' Alg.',I2,' Alg.',I2
     &,' Alg.',I2,' Alg.',I2,' Alg.',I2,' Alg.',I2
     &,' Alg.',I2,' Alg.',I2,' Alg.',I2,' Alg.',I2
     &,' Alg.',I2,' Alg.',I2,' Alg.',I2,' Alg.',I2 )
 2000 format ('No.   n   Conv      f^*        |f^*-fa|
     &        '||g^*||      Nf1  Nf2',
     &        '   r /  n    #   p   ',
     &        '   Variables x_i^*',/,
     &        108('-'))
```

```
 2001 FORMAT(' Pr# ',I2,': Fopt = ',e12.5,' nfe = ',i7,i8,i9,
     &        i3,' / ',i4,'  ',A1,' ',1f5.2,'s')
 2002 FORMAT(/,' Summary:',/,'   nfe_tot = ',I9,';  nfe_ave = ',i5,
     &         ';  # failures = ',i3,';  ttime = ',1f7.2,'s',//
     &         '   Gradient evaluations = ',i12,/,
     &         '   Normalized cost      = ',i12,/)
 3000 FORMAT (/,1X,' UNCONSTRAINED AND CONSTRAINED OPTIMIZATION VIA MULT
     &I-ALGORITHM IMPLEMENTATION
     &',//,
     & '      Copyright (c) June 2000 by Manie Bolton and Albert Groenwo
     &ld',//,
     & '                        All Commercial Rights Reserved ',/,
     & '               Use of this program for purposes other than',/,
     & '                 EDUCATION, RESEARCH and DEMONSTRATION',/,
     & '                        is Unprofessional and Illegal',//)
3467  format('Pr. = ',I2,' |',1f4.1,' ',1f4.1,' |',1f4.1,' ',1f4.1
     &,' |',1f4.1,' ',1f4.1,' |',1f4.1,' ',1f4.1,' |',1f4.1,' ',1f4.1
     &,' |',1f4.1,' ',1f4.1,' |',1f4.1,' ',1f4.1,' |',1f4.1,' ',1f4.1
     &,' |',1f4.1,' ',1f4.1,' |',1f4.1,' ',1f4.1,' |',1f4.1,' ',1f4.1
     &,' |',1f4.1,' ',1f4.1,' |',1f4.1,' ',1f4.1,' |',1f4.1,' ',1f4.1
     &,' |',1f4.1,' ',1f4.1)
 5001 FORMAT(' Pr# ',I2,' Nr# ',I2,' r# ',I3,' sl# ',I3,' Alg# ',I2
     &,': Fopt = ',e12.5,
     &' nfe = ',i7,i8,' qnr = ',f7.4,' x = ',10e14.5)
C ********************** FORMATS FOR WRITE *************************** C
      END
c ********************************************************************** C
C                                                                        C
C                     FUNCTION:CALCULATE THE SNYMAN-FATTI                C
C                     STOPPING CRITERIA CONVERGENCE PROBABILITY          C
C                                                                        C
c ********************************************************************** C
      FUNCTION CONVPROB(j,nr)
      IMPLICIT REAL*8 (A-H,O-Z)
      DATA ONE /1.D0/
      na=1  ! na = beta distribution B(a,b) parameter a
      nb=5  ! nb = beta distribution B(a,b) parameter b
      CONVPROB=1.d0
      DO K=0,j-1
      CONVPROB=CONVPROB*(2*j-nr+nb-1-K)/(2*J+nb+na-1-K)
      ENDDO
      CONVPROB=1.D0-CONVPROB
      RETURN
      END
c ********************************************************************** C
c                                                                        C
      subroutine shutdown( nproc, tids )
c                                                                        C
```

```
c ********************************************************************** C
      integer nproc, tids(*)
      do 10 i=1, nproc
         call pvmfkill( tids(i), info )         ! KILL ALL SPAWNED TASKS
  10  continue
      call pvmfexit( info )                              !  KILL MYSELF
  stop
      return
      end
C ********************************************************************** C
C                                                                        C
      SUBROUTINE CHOOSEMETHOD(METHOD,probil)
C                                                                        C
C     DETERMINES WHICH OPTIMIZATION ALGORITH WILL BE USED                C
C                                                                        C
C ********************************************************************** C
      IMPLICIT REAL*8 (A-H,O-Z),INTEGER(I-N)
      double precision probil(15),sumprob(15)
      dimension nalgorithm(15)
      integer ntimesalg(15)
      data nalgorithm /1,3,5,6,10,14,15,16,17,19,20,21,25,26,27/
C
        do kim=1,15
         if(kim.eq.1) then
         sumprob(1)=probil(1)
         else
         sumprob(kim)=sumprob(kim-1)+probil(kim)
         endif
        enddo
        if (sumprob(15).gt.1.000000001d0.or.sumprob(15).lt.0.989d0) then
        write(*,*) 'prob.gt.1',sumprob(15)
                write(*,*) 'sumprob',(sumprob(kl),kl=1,15)
        stop
        endif
    CALL RANMAR(P,1)
    ialg=0
 10         ialg=ialg+1
    IF (P.LE.sumprob(ialg)) THEN
            METHOD=nalgorithm(ialg)
    ELSE
    GOTO 10
    ENDIF
      return
      end
C ******************************** END ******************************** C
C ********************************************************************** C
```

F.4 Slave program for parallel optimization infrastructure

```
C ********************************************************************** C
C ********************************************************************** C
C                                                                        C
C       SLAVE PROGRAM FOR PARALLEL OPTIMIZATION BY H.P.J.BOLTON          C
C                                                                        C
C ********************************************************************** C
C ********************************************************************** C
      PROGRAM SLAVE
c ************************ STANDARD DECLARATIONS ********************** C
      IMPLICIT REAL*8(A-H,O-Z),INTEGER(I-N)
      include '../include/fpvm3.h'
      integer  info, mytid, mtid, msgtype, me
      integer  tids(128)
c ************************ STANDARD DECLARATIONS ********************** C
      INCLUDE 'params.inc'
      DOUBLE PRECISION X(MAXSIZ),XMIN(MAXSIZ)
      DOUBLE PRECISION BL(MAXSIZ),BU(MAXSIZ),GF(MAXSIZ)
      COMMON /KGRADEVAL/ NGRADEVAL
      COMMON /KFUNCEVAL/ NFUNCEVAL
      COMMON /PRDATA/ NPROB
      COMMON /DUMVAL/ DUMMY
      COMMON /ntimess/ jj
      COMMON /nslavess/ me
C
      call pvmfmytid( mytid )     ! GET SLAVE ID
      call pvmfparent( mtid )     ! GET MASTER ID
C
   1  continue
      NGRADEVAL=0
      NFUNCEVAL=0
C *************** RECEIVE DATA FROM MASTER(mtid) ********************** C
      msgtype  = 1
      call pvmfrecv( mtid, msgtype, info )
C
      call pvmfunpack( INTEGER4,nproc       ,     1, 1, info )
      call pvmfunpack( INTEGER4,tids        , nproc, 1, info )
      call pvmfunpack( INTEGER4,iseed       ,     1, 1, info )
      call pvmfunpack( INTEGER4,jseed       ,     1, 1, info )
      call pvmfunpack( INTEGER4,n           ,     1, 1, info )
      call pvmfunpack( INTEGER4,nprob       ,     1, 1, info )
      call pvmfunpack( INTEGER4,ni          ,     1, 1, info )
      call pvmfunpack( INTEGER4,ne          ,     1, 1, info )
      call pvmfunpack( INTEGER4,j           ,     1, 1, info )
      call pvmfunpack( REAL8,fmin           ,     1, 1, info )
```

```
      call pvmfunpack( REAL8,bl              ,    20, 1, info )
      call pvmfunpack( REAL8,bu              ,    20, 1, info )
      call pvmfunpack( REAL8,xmin            ,    20, 1, info )
      call pvmfunpack( REAL8,ndoprob         ,     1, 1, info )
      call pvmfunpack( INTEGER4,jj           ,     1, 1, info )
      call pvmfunpack( REAL8,x               ,    20, 1, info )
      call pvmfunpack( INTEGER4,METHOD       ,     1, 1, info )
C *************** RECEIVE DATA FROM MASTER(mtid) ********************** C
c  ************** DETERMINE WHICH SLAVE AM I (1 - nproc) ************* C
      DO I=1, nproc
         IF (tids(I).EQ.mytid) me = I
      ENDDO
c  ************** DETERMINE WHICH SLAVE AM I (1 - nproc) ************* C
C ***************** INITIALIZE THE RANDOM NUMBER GENERATOR *********** C
      ISEED=ISEED+me
      IF (ISEED.GT.31328) THEN
      ISEED=ISEED-31328
      ENDIF
      JSEED=JSEED+me
      IF (JSEED.GT.30081) THEN
      JSEED=JSEED-30081
      ENDIF
      CALL RMARIN(ISEED,JSEED)
C ***************** INITIALIZE THE RANDOM NUMBER GENERATOR *********** C
C
C ############## PERFORM OPTIMIZATION WITH CHOSEN METHOD ############## C
      call optimize(METHOD,NI,NE,n,f,x,gf,j,bl,bu,fmin,xmin)
C ############## PERFORM OPTIMIZATION WITH CHOSEN METHOD ############## C
C
C ********************** SEND RESULTS BACK TO MASTER ***************** C
      call pvmfinitsend( PVMDEFAULT, info )
C
      call pvmfpack( INTEGER4,me             ,     1, 1, info )
      call pvmfpack( INTEGER4,tids(me)       ,     1, 1, info )
      call pvmfpack( INTEGER4,NGRADEVAL      ,     1, 1, info )
      call pvmfpack( INTEGER4,NFUNCEVAL      ,     1, 1, info )
      call pvmfpack( REAL8, x                ,    20, 1, info )
      call pvmfpack( REAL8, f                ,     1, 1, info )
      call pvmfpack( REAL8, dummy            ,     1, 1, info )
      call pvmfpack( INTEGER4,method         ,     1, 1, info )
C
      msgtype  = 2
      call pvmfsend( mtid, msgtype, info )
C
C ********************** SEND RESULTS BACK TO MASTER ***************** C
      GOTO 1
      call pvmfexit(info)                    !  leave PVM before exiting
      STOP
```

```
      END
C ******************************** END ******************************** C
C ********************************************************************** C
```