SWEDISH DEFENCE
RESEARCH AGENCY

FOI-R--1593--SE
March 2005
ISSN 1650-1942
Technical report

Hanna Gothäll, Rune Westin

Evaluation of Four Global Optimisation Techniques (ASSA, DE, NA, Tabu Search) as Applied to Anechoic Coating Design and Inverse Problem Uncertainty Estimation

Systems Technology
SE-172 90 Stockholm

SWEDISH DEFENCE RESEARCH AGENCY

Systems Technology
SE-172 90 Stockholm

FOI-R--1593--SE

March 2005

ISSN 1650-1942

Technical report

Hanna Gothäll, Rune Westin

Evaluation of Four Global Optimisation Techniques (ASSA, DE, NA, Tabu Search) as Applied to Anechoic Coating Design and Inverse Problem Uncertainty Estimation

Issuing organization FOI – Swedish Defence Research Agency Systems Technology SE-172 90 Stockholm	**Report number, ISRN** FOI-R--1593--SE	**Report type** Technical report
	Research area code 43 Underwater Sensors	
	Month year March 2005	**Project no.** E6058
	Sub area code 43 Underwater Sensors	
	Sub area code 2	
Author/s (editor/s) Hanna Gothäll Rune Westin	**Project manager** Henrik Classon	
	Approved by Monica Dahlén	
	Sponsoring agency Swedish armed forces	
	Scientifically and technically responsible Sven Ivansson	

Report title

Evaluation of Four Global Optimisation Techniques (ASSA, DE, NA, Tabu Search) as Applied to Anechoic Coating Design and Inverse Problem Uncertainty Estimation

Abstract (not more than 200 words)

During the last ten to fifteen years, simulated annealing and genetic algorithms have become routine tools in the field of underwater acoustics for solving difficult optimisation and inverse problems. However, other global optimisation methods have recently been introduced, some of which have been reported to outperform the previous ones.

In the present report, four such more modern global optimisation techniques are tested and compared: Adaptive Simplex Simulated Annealing (ASSA), Differential Evolution (DE), Neighbourhood Algorithm (NA), and Enhanced Continuous Tabu Search (ECTS). The techniques have been tested on synthetic optimisation problems and applied to the design of Alberich anechoic coatings. ASSA and DE performed best of the four algorithms in the synthetic test problems, as well as in the coating design problem. For the other two algorithms, NA and ECTS, further research is desired in order to improve their exploring capabilities.

In the context of inverse problems, a solution appraisal stage is important, and an evaluation of a recently developed method for that purpose is reported. A Bayesian inversion problem was formulated concerning the Alberich anechoic coatings, and the optimisation algorithms were applied to obtain least-squares solutions. An extension of NA was then used to get an indication of the variable estimate uncertainties, by resampling the obtained search ensembles. This extension, called NA-Bayes, was found to be useful tool for the solution appraisal stage, provided that the search domain is well sampled by the optimisation technique used.

Keywords

Adaptive Simplex Simulated Annealing, Differential Evolution, Neighbourhood Algorithm, Enhanced Continuous Tabu Search, Neighbourhood Algorithm – Bayes, anechoic coatings

Further bibliographic information	**Language** English
ISSN 1650-1942	**Pages** 85 p.
	Price acc. to pricelist

Utgivare Totalförsvarets Forskningsinstitut - FOI Systemteknik 172 90 Stockholm	**Rapportnummer, ISRN** FOI-R--1593--SE	**Klassificering** Teknisk rapport
	Forskningsområde 4. Ledning, informationsteknik och sensorer	
	Månad, år Mars 2005	**Projektnummer** E6058
	Delområde 43 Undervattenssensorer	
	Delområde 2	
Författare/redaktör Hanna Gothäll Rune Westin	**Projektledare** Henrik Classon	
	Godkänd av Monica Dahlén	
	Uppdragsgivare/kundbeteckning Försvarsmakten	
	Tekniskt och/eller vetenskapligt ansvarig Sven Ivansson	

Rapportens titel (i översättning)

Utvärdering av fyra globala optimeringstekniker (ASSA, DE, NA, Tabu Search) tillämpade på design av ekodämpande skikt och osäkerhetsuppskattning för inversproblem

Sammanfattning (högst 200 ord)

Under de senaste tio till femton åren har simulated annealing och genetiska algoritmer blivit rutinverktyg inom undervattensakustiken för att lösa svåra optimerings- och inversproblem. På senare tid har dock nyare optimeringsalgoritmer introducerats. Dessa algoritmer har i vissa fall rapporterats prestera betydligt bättre än sina föregångare.

I denna rapport testas och jämförs fyra sådana moderna globala optimeringsalgoritmer: Adaptive Simplex Simulated Annealing (ASSA), Differential Evolution (DE), Neighbourhood Algorithm (NA), och Enhanced Continuous Tabu Search (ECTS). Teknikerna testades på syntetiska optimeringsproblem och tillämpades på design av ekodämpande skikt av alberichtyp. ASSA och DE presterade bäst av de fyra algoritmerna för testproblemen såväl som för designproblemet. De andra två algoritmerna, NA och ECTS, behöver utvecklas ytterligare för att deras förmågor till global utforskning ska förbättras.

I inversproblemssammanhang är det viktigt med en bedömning av osäkerheten i den erhållna lösningen. För detta ändamål har en nyligen utvecklad metod prövats. Ett bayesianskt inversproblem formulerades baserat på de ekodämpande skikten och optimeringsalgoritmerna användes för att erhålla minsta-kvadratlösningar. En utvidgning av NA användes sedan för att få en uppfattning om osäkerheten i variabelskattningen, genom att samplera om de erhållna sökensemblerna. Denna utvidgning, vid namn NA-Bayes, visade sig vara ett användbart verktyg för bedömningen av lösningens trovärdighet. En förutsättning är dock att den använda optimeringsalgoritmen har samplerat sökrymden väl.

Nyckelord

Adaptive Simplex Simulated Annealing, Differential Evolution, Neighbourhood Algorithm, Enhanced Continuous Tabu Search, Neighbourhood Algorithm – Bayes, ekodämpande skikt

Övriga bibliografiska uppgifter	**Språk** Engelska
ISSN 1650-1942	**Antal sidor:** 85 s.
Distribution enligt missiv	**Pris: Enligt prislista**

Abstract

During the last ten to fifteen years, simulated annealing and genetic algorithms have become routine tools in the field of underwater acoustics for solving difficult optimisation and inverse problems. However, other global optimisation methods have recently been introduced, some of which have been reported to outperform the previous ones.

In the present report, four such more modern global optimisation techniques are tested and compared: Adaptive Simplex Simulated Annealing (ASSA), Differential Evolution (DE), Neighbourhood Algorithm (NA), and Enhanced Continuous Tabu Search (ECTS). The techniques have been tested on synthetic optimisation problems and applied to the design of Alberich anechoic coatings. ASSA and DE performed best of the four algorithms in the synthetic test problems, as well as in the coating design problem. For the other two algorithms, NA and ECTS, further research is desired in order to improve their exploring capabilities.

In the context of inverse problems, a solution appraisal stage is important, and an evaluation of a recently developed method for that purpose is reported. A Bayesian inversion problem was formulated concerning the Alberich anechoic coatings, and the optimisation algorithms were applied to obtain least-squares solutions. An extension of NA was then used to assess the variable estimate uncertainties, by resampling the obtained search ensembles. This extension, called NA-Bayes, was found to be useful tool for the solution appraisal stage, provided that the search domain is well sampled by the optimisation technique used.

Preface

This report contains the Master's Thesis work of the authors, required for the M.Sc. degree at the Marcus Wallenberg Laboratory for Sound and Vibration Research (MWL) at the Royal Institute of Technology (KTH). The work was performed at the division of Systems Technology at the Swedish Defence Research Agency (FOI), as part of ongoing underwater acoustics. It was supported by three specific research projects: Underwater signatures, naval surveillance systems, and marine tactical support systems.

Chapters 3 and 5 have been written by Hanna Gothäll while chapters 4 and 6 have been written by Rune Westin. The remaining chapters have been written jointly by Hanna Gothäll and Rune Westin. This division of the authorship perfectly reflects the division of the work.

Acknowledgements

The authors would like to thank the following people who have in some way or another contributed to this work:

- Our supervisor *Sven Ivansson* for inspiration, support and the opportunity to visit the Scandinavian Symposium on Physical Acoustics in Ustaoset, Norway
- Our examiner *Ilkka Karasalo* for support and the opportunity to visit the Scandinavian Symposium on Physical Acoustics in Ustaoset, Norway
- All the people at *Numerical Wave Propagation and Analysis* and *Signal Processing*

Hanna would like to thank:

- *Stan E. Dosso* for making the ASSA code available and for the enlightening dialog
- *Malcolm Sambridge* for making the NA code available and for the figure 9.2

Rune would like to thank:

- *Rachid Chelouah* and *Patrick Siarry* for the help on the ECTS promising list
- *Gade Pandu Rangaiah* and *Yong Sing Teh* for further help on ECTS
- *Gerd Eriksson* for introducing me to the exciting topic of optimisation

And last, but not least, we would like to thank *Rikard Gothäll* and *Malin Sandström* for encouragement, support, and LaTeX help

Contents

Chapter 1

Introduction

Hanna Gothäll and Rune Westin

The ability to solve optimisation problems improves as computers get more powerful. However, the brute force method to exhaustively search the parameter space is generally out of the question, even with the computing power available today. There is a need for algorithms that effectively search for optima. Search methods that make use of local linear approximations can be used for solving linear or weakly non-linear problems.

A difficulty arises, however, when the global optimum is desired, while the function to be optimised has many local optima. Derivative-free global search methods are needed to handle difficult non-linear problems. The algorithms do not know if an optimum found is the global optimum. What the algorithms need is the ability to move their attentions from regions with good values in order to search globally for regions with potentially better values.

In this study, four modern global optimisation techniques have been tested and compared. These four techniques are: Adaptive Simplex Simulated Annealing (chapter 3), Differential Evolution (chapter 4), Neighbourhood Algorithm (chapter 5), and Enhanced Continuous Tabu Search (chapter 6). They have been tested on the synthetic test problems described in chapter 2, and the different methods are compared in chapter 7. The optimisation techniques have also been applied to the non-linear problem of optimising the design of Alberich anechoic coatings (chapter 8).

Furthermore, an inversion problem, based on the Alberich anechoic coatings, has been studied. In chapter 9, the inversion problem and Neighbourhood Algorithm – Bayes are described. This algorithm is used to assess variable estimate uncertainties, by resampling search ensembles. The ensembles are created by the four optimisation techniques when solving inversion problems in the least-squares sense.

1.1 Definitions

When studying what is written on the subject of global optimisation techniques, it is apparent that there are wide variations in the nomenclatures used in different fields of science and for different types of algorithms. Many of the central concepts in global optimisation are described in different ways depending on what paper, book or web page is consulted.

The intention is to use a consistent nomenclature throughout this report, implying that some common terms for certain techniques are replaced. Some of the terms used are defined below.

1.1.1 Models in the Search Domain

The optimisation involves the determination of a vector $\mathbf{m}$ in a *search domain* which is assumed to be a subset of Euclidian space with dimension N. Each vector $\mathbf{m}$ is called a *model.* The elements of $\mathbf{m}$ are called *model variables* and are denoted m_i, where $i = 1, \ldots, N$.

1.1.2 Objective Function

The objective of a global optimisation technique is to find the lowest (or in some cases the highest) possible value of a function $f(\mathbf{m})$. This function $f(\mathbf{m})$, uniquely associating each possible model $\mathbf{m}$ with a real, scalar value, is in this report called an *objective function.* The associated value is in turn called the *objective function value.*

The purpose of the optimisation is to find the model, in the search domain, with the lowest (or highest) objective function value.

All optimisations in the present report concern minimisation.

Chapter 2

Synthetic Test Problems

Hanna Gothäll and Rune Westin

The performance of an optimisation technique often depends greatly on the tuning of control parameters, such as the population size for Differential Evolution or the cooling factor for Adaptive Simplex Simulated Annealing. It is desired to have guidelines for choosing control parameters values when optimising applied problems where the global optima are not known. Therefore, five test problems are used to evaluate the different optimisation techniques. These test problems are functions of two or six variables.

Statistics are gathered in order to evaluate the reliability of the optimisation techniques for different values of the control parameters. The reliability of each technique is measured by estimating the probability of success in finding the global optima for the different test problems. Since the techniques are intended to find a model "close enough" to the global optimum rather than the global optimum itself, a limit for "close enough" for success has to be defined. In the tests, the following criterion from [1] is used:

$$\begin{aligned}&f(\mathbf{m}_{\text{low}}) < f_{\text{limit}} \quad \text{where}\\ &f_{\text{limit}} = f(\mathbf{m}_{\text{opt}}) + \epsilon_{\text{rel}} \cdot |f(\mathbf{m}_{\text{opt}})| + \epsilon_{\text{abs}}\end{aligned} \tag{2.1}$$

and $\mathbf{m}_{\text{low}}$ is the model with the lowest objective function value found, $\mathbf{m}_{\text{opt}}$ is the analytical global optimum, $\epsilon_{\text{rel}} = 10^{-4}$ and $\epsilon_{\text{abs}} = 10^{-6}$. This gives the objective function value limit f_{limit} that has to be reached *for the optimisation to succeed.*

2.1 Phases

A *phase* is defined as a number of optimisation runs with a particular control parameter value setup for which a series of statistics is collected as described below. The statistics give information on the suitability of this particular setup.

2.1.1 Estimated Probability

For a certain phase, let P be the probability of success in finding the global optimum (as defined above). The stochastic variable X is defined as the number of successful optimisations during n_{run} runs. *The probability of success for a phase is estimated by the outcome of* $\frac{X}{n_{run}}$.

The stochastic variable X is binomially distributed and the standard deviation σ of $\frac{X}{n_{run}}$ is

$$\sigma = \sqrt{\frac{P \cdot (1-P)}{n_{run}}}, \tag{2.2}$$

providing an indication of the uncertainty of the estimate. The maximum σ appears at $P =$

0.5. For example, the maximum σ is $5 \cdot 10^{-3}$ for $n_{run} = 10000$ and $1.6 \cdot 10^{-2}$ for $n_{run} = 1000$.

P can also be considered as a function of n_{eval}, the number of objective function evaluations performed. This can be emphasised by writing $P(n_{eval})$. At predetermined values for n_{eval} in the optimisations, the outcome $x(n_{eval})$ of $X(n_{eval})$ is observed, and $P(n_{eval})$ for the corresponding phase is estimated by

$$P_{\text{est}}(n_{eval}) = \frac{x(n_{eval})}{n_{run}}. \tag{2.3}$$

2.1.2 Mean Deviation from Optimum

In addition to the estimated probability of success, the best models found at different values of n_{eval} are observed and denoted $\mathbf{m}_{\text{low}}(n_{eval})$. The mean deviation from the global optimum can then be calculated as

$$MD(n_{eval}) = \\ = \frac{1}{n_{run}} \cdot \sum_{1}^{n_{run}} (f(\mathbf{m}_{\text{low}}(n_{eval})) - f(\mathbf{m}_{\text{opt}})). \tag{2.4}$$

2.2 Test Problems

2.2.1 Mexican Hat

This test function has also been studied in [2]. The objective function, illustrated in figure 2.1, has a global minimum value at the origin. The other local minima are circles around the origin and the objective function values of these minima increase slowly with increasing radius.

$$f(x_1, x_2) = -0.5 + \frac{\sin^2(\sqrt{x_1^2 + x_2^2}) - 0.5}{(1 + 0.001(x_1^2 + x_2^2))^2} \tag{2.5}$$

- Number of variables: $N = 2$

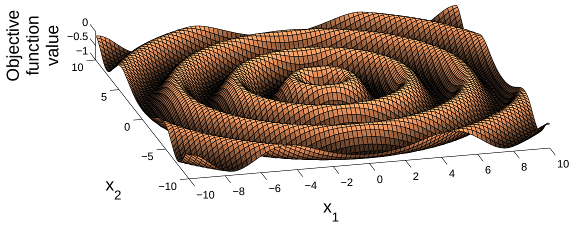


Figure 2.1: The Mexican Hat test problem on the interval [-10,10] on both axes.

- Search domain: $-100 \leq x_i \leq 100, \quad i = 1, 2$
- Local minima: Several
- Global minima: One
  $\mathbf{m}_{\text{opt}} = (0, 0), \quad f(\mathbf{m}_{\text{opt}}) = -1$
- Limit for success[1]: $f_{\text{limit}} = -0.99$

There are actually two regions with objective function values below the limit. One is the region immediately around the global minimum at the origin. The other region is the bottom of the first circular "valley".

2.2.2 Fallat-Dosso

This test function is taken from [3] and bears a closer resemblance to real inversion problems than the other test functions. There are many local minima and the global minimum is situated at the origin (see figure 2.2).

[1]In Mexican hat, the limit given in equation 2.1 was exchanged for the limit used in [2] to make comparisons possible.

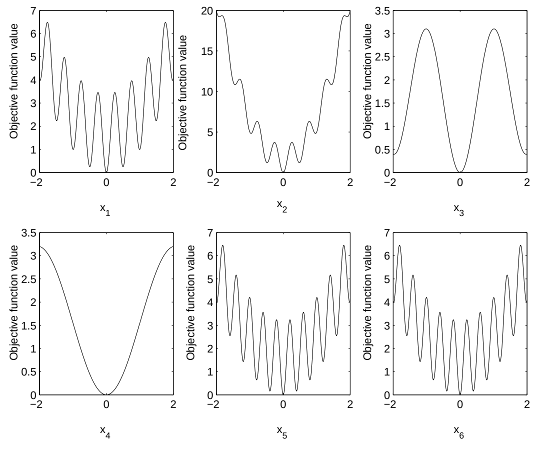


Figure 2.2: The test problem Fallat-Dosso. The objective function value plotted against each of the six variables. The scale for all variables is [-2,2].

$$
\begin{aligned}
f(x_1,&x_2,x_3,x_4,x_5,x_6) = \\
&=4.8 + x_1^2 + 5x_2^2 + 0.1x_3^2 \\
&\quad + 0.05x_4^2 + x_5^2 + x_6^2 \\
&\quad - 0.3\cos(4\pi(x_1 - x_2)) \\
&\quad - 1.4\cos(4\pi(x_1 + x_2)) \qquad (2.6)\\
&\quad - 0.5\cos(10\pi(0.05x_4 - 0.01x_3)) \\
&\quad - 1.0\cos(10\pi(0.05x_4 + 0.1x_3)) \\
&\quad - 0.25\cos(5\pi(x_5 - x_6)) \\
&\quad - 1.35\cos(5\pi(x_5 + x_6))
\end{aligned}
$$

- Number of variables: $N = 6$
- Search domain: $-2 \leq x_i \leq 2, \quad i = 1,\dots,6$
- Local minima: Several
- Global minima: One
  $\mathbf{m}_{\text{opt}} = (0,0,0,0,0,0), \quad f(\mathbf{m}_{\text{opt}}) = 0$
- Limit for success[2]: $f_{\text{limit}} = 1 \cdot 10^{-5}$

[2]In Fallat-Dosso, the limit given in equation 2.1 was exchanged for the limit used in [3] to make comparisons possible.

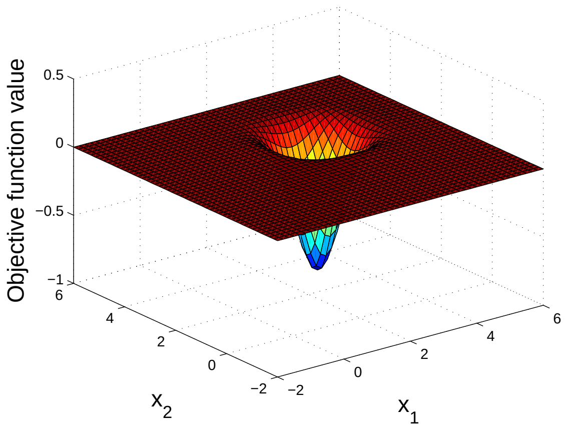


Figure 2.3: The Easom test problem on the interval [-2,6] on both axes. Note that the search domain spans the interval [-100,100] in both dimensions.

2.2.3 Easom

Easom has been studied in [1] and features a function that is mostly uniform, except for a "well" around $\mathbf{m} = (\pi,\pi)$ as illustrated in figure 2.3. The flat area extends far outside what is shown in the figure, making the well a very small part of the search domain. There are extremely small variations in the objective function value in the "flat" surface. The magnitude of these variations are controlled by the factor e^{-r^2} where $r = \sqrt{(\pi - x_1)^2 + (\pi - x_2)^2}$ is the distance from (π,π) as seen in equation (2.7). For example, at distances $r > 5$ from (π,π), $|f(x_1,x_2)| < 10^{-10}$.

$$
\begin{aligned}
f(x_1,x_2) = &- \cos(x_1)\cos(x_2) \\
&\cdot e^{-((x_1-\pi)^2+(x_2-\pi)^2)}
\end{aligned} \qquad (2.7)
$$

- Number of variables: $N = 2$
- Search domain: $-100 \leq x_i \leq 100, \quad i = 1,2$
- Local minima: Several, but essentially insignificant. Outside a radius of 1.5π from the global minimum, $|f(x_1,x_2)| < 10^{-10}$.

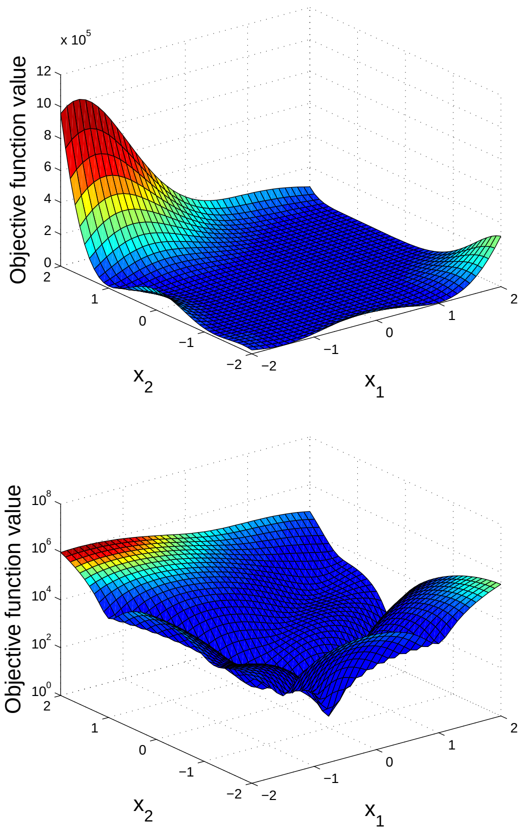


Figure 2.4: The Goldstein-Price test problem on the interval [-2,2] on both axes. Note the large maximum value of the objective function value. The scales of the vertical axis in the lower panel is logarithmic.

- Global minima: One
  $\mathbf{m}_{\text{opt}} = (\pi, \pi), \quad f(\mathbf{m}_{\text{opt}}) = -1$
- Limit for success: $f_{\text{limit}} = -0.99989$

2.2.4 Goldstein-Price

This test function has also been studied in [1]. It has only three local optima with smooth surfaces in between as illustrated in figure 2.4.

$$\begin{aligned} f(x_1, x_2) = &[1 + (x_1 + x_2 + 1)^2 \\ &\cdot (19 - 14x_1 + 3x_1^2 - 14x_2 \\ &+ 6x_1x_2 + 3x_2^2)] \\ &\cdot [30 + (2x_1 - 3x_2)^2 \\ &\cdot (18 - 32x_1 + 12x_1^2 + 48x_2 \\ &- 36x_1x_2 + 27x_2^2)] \end{aligned} \tag{2.8}$$

- Number of variables: $N = 2$
- Search domain: $-2 \leq x_i \leq 2, \quad i = 1, 2$
- Local minima: Four
  $f(1.2, 0.8) = 840$,
  $f(1.8, 0.2) = 84$,
  $f(-0.6, 0.8) = 30$,
  $f(0, -1) = 3$
- Global minima: One
  $\mathbf{m}_{\text{opt}} = (0, -1), \quad f(\mathbf{m}_{\text{opt}}) = 3$
- Limit for success: $f_{\text{limit}} = 3.000301$

2.2.5 Shubert

Shubert has also been studied in [1] and has 18 global minima in contrast to the other test functions. In figure 2.5, a ninth of the search domain is shown and it can be seen that the surface of the objective function is very rugged. The surface shown in figure 2.5 is repeated throughout the search domain.

$$\begin{aligned} f(x_1, x_2) = &\left\{ \sum_{j=1}^{5} j \cos((j+1)x_1 + j) \right\} \\ &\cdot \left\{ \sum_{j=1}^{5} j \cos((j+1)x_2 + j) \right\} \end{aligned} \tag{2.9}$$

- Number of variables: $N = 2$
- Search domain: $-10 \leq x_i \leq 10, \quad i = 1, 2$

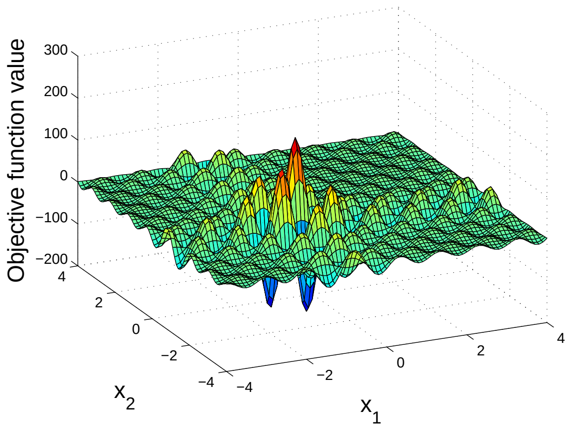


Figure 2.5: The test problem Shubert. Here, only a ninth is shown for clarity. The interval is [-4,4] on both axes.

- Local minima: 760
- Global minima: 18
  $f(\mathbf{m}_{\text{opt}}) \approx -186,7309$
- Limit for success: $f_{\text{limit}} = -186.7122$

Chapter 3

Adaptive Simplex Simulated Annealing

Hanna Gothäll

3.1 Background

The hybrid inverse methods are relatively new alternatives to local and global inversion algorithms. The hybrids combine a local and a global algorithm and intend to retain the advantages of both, but get rid of the disadvantages. The goal is to effectively move downhill but avoid becoming trapped in local minima. The Adaptive Simplex Simulated Annealing (ASSA) algorithm is based on the two search algorithms Fast Simulated Annealing (FSA) and Downhill Simplex (DHS).

3.1.1 Simulated Annealing

Simulated Annealing (SA) is a global search algorithm which only remembers one model: the one it is working with at the moment. The model parameters are perturbed randomly one by one with perturbations drawn from some probability distribution, in order to walk to the next model and see if that one is better. The idea is to escape local minima and continue to search for a better model closer to the global optimum. The new perturbed model is always accepted if the objective function value f is decreased (criterion 1), otherwise the model is accepted with a probability drawn from the Boltzmann distribution

$$P(\Delta f) = e^{-\Delta f/T} \tag{3.1}$$

where Δf is the difference in objective function values for the new and the old model (criterion 2). The above formula is part of the Metropolis algorithm. T is the temperature, a control parameter which is decreased successively. The temperature is decreased in order to have a global search in the beginning and a more local search in the end.[1]

The annealing schedule is defined by the control parameters for SA. The parameters are the initial temperature, the cooling rate, the number of temperature steps and the number of perturbations per temperature step.

After all the temperature steps are run through, quenching takes place to ensure that the bottom of the closest minimum is reached. In this procedure, the temperature is set to zero such that only values that decrease f are accepted.

[1]The terminology above is often used in the SA context because of the analogy with thermodynamics. Usually E (energy) is used in the SA context, but here f is used.

Fast Simulated Annealing (FSA)

In Fast Simulated Annealing (FSA), the perturbations are drawn from the Cauchy distribution about the current parameter values. The width of the distribution is decreased linearly with temperature to accelerate convergence. This is why the algorithm is named *fast*. Large perturbations are required in the beginning to get a wider search, but the search gets more and more local when the temperature decreases. An appropriate annealing schedule is required for FSA to be effective.

Shortcomings of SA and FSA are their lack of memory and their poor ability to move quickly downhill. For example, a good model can be visited in an early stage of the search and never be revisited, since it has not been stored in any memory.

Table 3.1: A summary of the FSA algorithm

1. Choose a model at random in parameter space;
2. Perturb the model parameters at random with perturbations drawn from the Cauchy distribution;
3. Check if the model has lowered the objective function value or is accepted by the Boltzmann probability (see equation 3.1). If not accepted, go back to 2;
4. Decrease the temperature. Go back to 2 unless all iterations have been made;
5. Set the temperature to zero and perform quenching for a certain number of iterations.

3.1.2 Downhill Simplex

Downhill Simplex (DHS) is a local derivative-free search algorithm which does not involve solving systems of equations. Nelder and Mead [4] were the first to formulate the algorithm.

DHS uses a simplex involving $N+1$ models $\mathbf{m}_{\text{low}}, ..., \mathbf{m}_{2^{\text{nd}}\text{high}}, \mathbf{m}_{\text{high}}$, where N is the dimension of parameter space (see figure 3.1(a) where $N = 2$), $\mathbf{m}_{\text{low}}$ is the model with the lowest objective function value, $\mathbf{m}_{2^{\text{nd}}\text{high}}$ is the model with the second highest objective function value and $\mathbf{m}_{\text{high}}$ is the model with the highest objective function value. $\mathbf{m}_{\text{high}}$ is first reflected through the simplex made by the N models with the lower objective function values, which is a line segment in two dimensions (see figure 3.1(b)). If the objective function value of the reflected model, $f(\mathbf{m}_{\text{refl}})$, lies between $f(\mathbf{m}_{\text{low}})$ and $f(\mathbf{m}_{2^{\text{nd}}\text{high}})$, $\mathbf{m}_{\text{refl}}$ replaces $\mathbf{m}_{\text{high}}$.

If $f(\mathbf{m}_{\text{refl}}) \leq f(\mathbf{m}_{\text{low}})$, $\mathbf{m}_{\text{high}}$ is reflected through the same simplex as before but the distance it is reflected is expanded by a factor two (see figure 3.1(c)) and a new model $\mathbf{m}_{\text{reflExp}}$ is created. This is a way to check if the models continue to get better in this particular direction as the objective function value of the reflected model indicated. The model of $\mathbf{m}_{\text{refl}}$ and $\mathbf{m}_{\text{reflExp}}$ with the lowest objective function value replaces $\mathbf{m}_{\text{high}}$.

If $f(\mathbf{m}_{\text{refl}})$ is higher than or equal to $f(\mathbf{m}_{\text{high}})$, the distance is instead contracted by a factor of two and the model $\mathbf{m}_{\text{con}}$ is created. The model then lies between $\mathbf{m}_{\text{high}}$ and the smaller simplex made of the N best points in the simplex with the $N+1$ models (see figure 3.1(d)). If $f(\mathbf{m}_{\text{con}}) < f(\mathbf{m}_{\text{high}})$, $\mathbf{m}_{\text{con}}$ replaces $\mathbf{m}_{\text{high}}$. Otherwise a multiple contraction is performed.

If $f(\mathbf{m}_{\text{refl}})$ is between $f(\mathbf{m}_{2^{\text{nd}}\text{high}})$ and $f(\mathbf{m}_{\text{high}})$, then the model with the highest objective function value is moved by a reflection and contraction to lie between the reflected

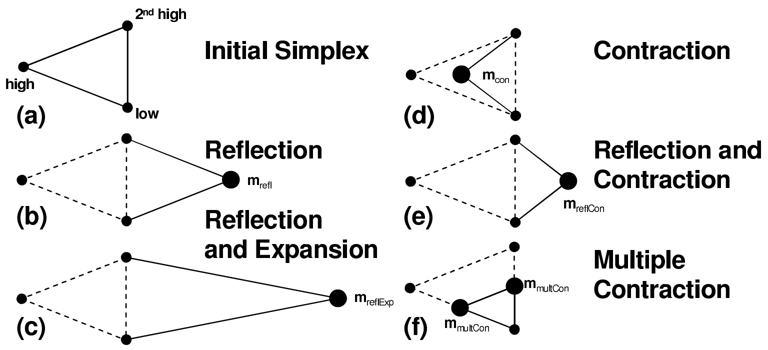


Figure 3.1: DHS steps in two dimensions

point and the simplex of the N best points, as shown in figure 3.1(e). If $f(\mathbf{m}_{\text{reflCon}}) < f(\mathbf{m}_{\text{high}})$, $\mathbf{m}_{\text{reflCon}}$ replaces $\mathbf{m}_{\text{high}}$. Otherwise a multiple contraction is performed.

In a multiple contraction, the N models with the highest objective function values in the simplex are moved towards the model with the lowest objective function value (see figure 3.1(f)).

The iterations (step 2 through step 6 in table 3.2) are repeated until the difference between $f(\mathbf{m}_{\text{high}})$ and $f(\mathbf{m}_{\text{low}})$ relative to their average is less than some tolerance ϵ:

$$\frac{f(\mathbf{m}_{\text{high}}) - f(\mathbf{m}_{\text{low}})}{(f(\mathbf{m}_{\text{high}}) + f(\mathbf{m}_{\text{low}}))/2} < \epsilon \qquad (3.2)$$

A summary of the above explanation of the algorithm can be found in table 3.2.

3.2 Description of the ASSA Algorithm

As stated by Dosso et al. [6], the adaptive simplex simulated annealing (ASSA) algorithm is based on perturbations that combine local DHS steps and a random component with the size of the random component for each parameter reduced in an adaptive manner based on recent search statistics, rather than in an arbitrary manner as in standard FSA.

The algorithm was published by Dosso, Wilmut and Lapinski [6] in the year 2001.

Before ASSA was developed, DHS and FSA had been combined by Fallat and Dosso into Simplex Simulated Annealing (SSA) [3], [7]. Hedar and Fukushima have also published an algorithm combining SA and DHS [8].

The description here is only a short summary of the algorithm, for a more thorough explanation, see Dosso et al. [6].

The ASSA algorithm is an adaptive combination of DHS and FSA. ASSA uses a simplex of models (N+1 models) instead of only one model as SA and FSA do (figure 3.2 (a)). This leads to an increase of the algorithm memory from one model to N+1 models, and the best model visited is always saved in the simplex.

After a DHS step (figure 3.2 (b)), all parameters in the new model are perturbed (figure 3.2 (c)) with a perturbation drawn from a temperature dependent Cauchy distribution (as in FSA). A DHS step can be a reflection, expansion or contraction; cf. step 2 through step 5 in table 3.2. If the perturbed model, for example $\mathbf{m}_{\text{reflPert}}$, is accepted by one of the two SA criteria, the simplex is updated (figure 3.2 (d)).

A summary of the ASSA algorithm is found

Table 3.2: A summary of the DHS algorithm (after [5])

1. Choose a simplex with $N + 1$ models at random in the parameter space (figure 3.1(a));

2. Reflect (figure 3.1(b)) the model with the highest objective function value in the simplex, $\mathbf{m}_{\text{high}}$. If the objective function value of the reflected model, $f(\mathbf{m}_{\text{refl}})$, fulfils:

   (a) $f(\mathbf{m}_{\text{low}}) < f(\mathbf{m}_{\text{refl}}) < f(\mathbf{m}_{2^{\text{nd}}\text{high}})$, replace $\mathbf{m}_{\text{high}}$ with $\mathbf{m}_{\text{refl}}$, go to 7;

   (b) $f(\mathbf{m}_{\text{refl}}) \leq f(\mathbf{m}_{\text{low}})$, go to 3;

   (c) $f(\mathbf{m}_{\text{refl}}) \geq f(\mathbf{m}_{\text{high}})$, go to 4;

   (d) $f(\mathbf{m}_{2^{\text{nd}}\text{high}}) < f(\mathbf{m}_{\text{refl}}) < f(\mathbf{m}_{\text{high}})$, go to 5;

3. Reflect $\mathbf{m}_{\text{high}}$ and expand by a factor two (figure 3.1(c)) to get $\mathbf{m}_{\text{reflExp}}$. The one of $\mathbf{m}_{\text{refl}}$ and $\mathbf{m}_{\text{reflExp}}$ with the lowest objective function value replaces $\mathbf{m}_{\text{high}}$. Go to 7;

4. Contract $\mathbf{m}_{\text{high}}$ by a factor two (figure 3.1(d)) to get $\mathbf{m}_{\text{con}}$. If $f(\mathbf{m}_{\text{con}}) < f(\mathbf{m}_{\text{high}})$ then replace $\mathbf{m}_{\text{high}}$ with $\mathbf{m}_{\text{con}}$ and go to 7, else go to 6;

5. Reflect $\mathbf{m}_{\text{high}}$ and contract by a factor two (figure 3.1(e)) to get $\mathbf{m}_{\text{reflCon}}$. If $f(\mathbf{m}_{\text{reflCon}}) < f(\mathbf{m}_{\text{high}})$ then replace $\mathbf{m}_{\text{high}}$ with $\mathbf{m}_{\text{reflCon}}$ and go to 7, else go to 6;

6. Perform a multiple contraction (figure 3.1(f)) around $\mathbf{m}_{\text{low}}$. Go to 7;

7. Continue from 2 until the stop criterion, equation (3.2), is fulfilled.

in tables 3.3 and 3.4. DHS steps including perturbations are performed repeatedly. A reflection (step 2) is possibly followed by an expansion (step 4) or a contraction (step 5), and the process is repeated starting with another reflection step. The temperature is reduced when enough models have been accepted at the current temperature (step 6 through step 7).

When perturbing a model, the width of the distribution for each parameter is adaptively scaled using a factor of $s > 1$ times the running average of the last S corresponding random perturbations for which the model was included in the simplex. This is done to be able to generate new perturbations with a reasonable probability of acceptance. The perturbations are also dependent on the temperature. They are at first large, but they decrease with time in order to have a more global search in the beginning and a more localised search in the end. In general, the larger the factor s, the slower but more thorough the parameter search [6]. Because of these perturbations from FSA, ASSA is able to climb up from local minima and is therefore an improvement in relation to DHS.

The temperature is decreased with a factor (β, see the control parameter section) until all iterations are done. The initial temperature, T_0, is chosen such that 90% of the models are accepted in the beginning of the run. A summary of the ASSA algorithm is found in tables 3.3 and 3.4.

Quenching was removed because it rarely improved the result if a good cooling rate was chosen (Dosso, personal communication).

Multiple contractions are not included in the original ASSA algorithm. However, see section 3.4.2.

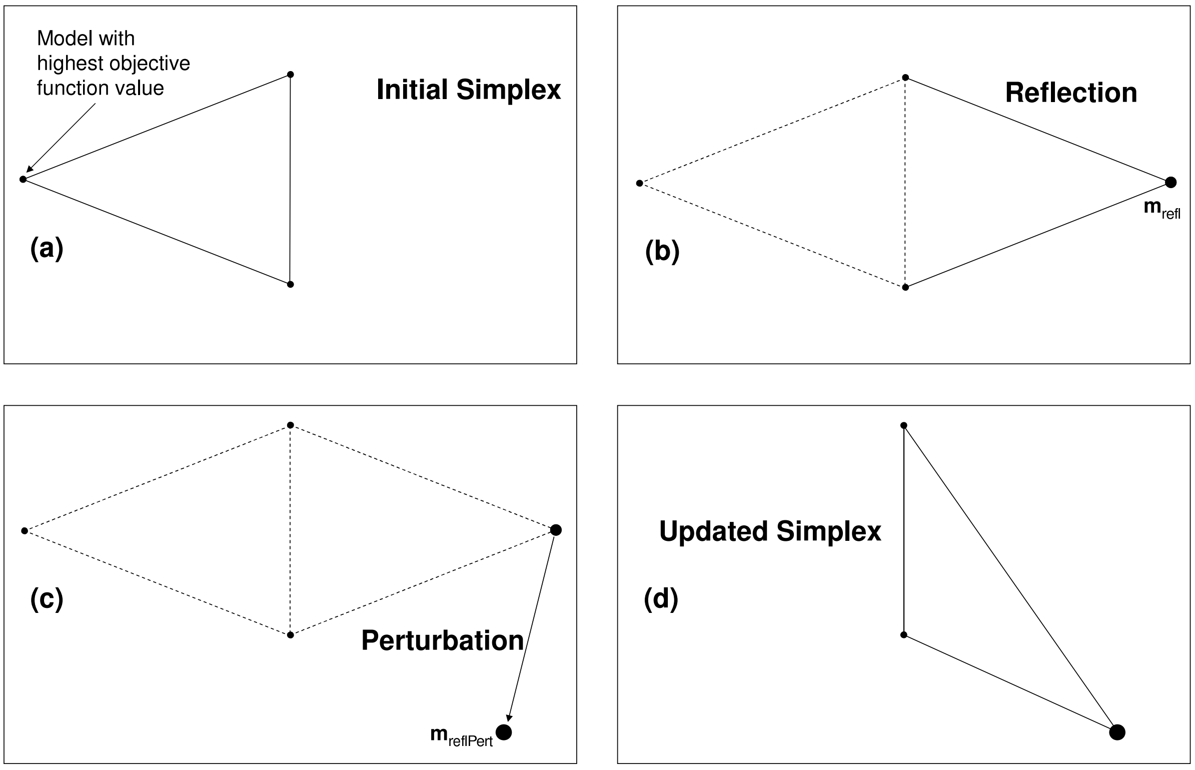

Model with highest objective function value
Initial Simplex
(a)
Reflection
$\mathbf{m}_{\text{refl}}$
(b)
(c)
Perturbation
$\mathbf{m}_{\text{reflPert}}$
Updated Simplex
(d)

Figure 3.2: Illustration of the replacement of one model in ASSA. (a) An initial simplex is chosen randomly in the search domain. (b) A DHS iteration is performed for the model with the highest objective function value in the initial simplex. In this case, a reflection is made. (c) The reflected model $\mathbf{m}_{\text{refl}}$ is perturbed with a Cauchy distributed perturbation and the model $\mathbf{m}_{\text{reflPert}}$ is created. (d) If the perturbed model, $\mathbf{m}_{\text{reflPert}}$, is accepted, the simplex is updated and the procedure continues.

Table 3.3: First part of a summary of the ASSA algorithm. Continued in table 3.4.

1. Randomly initialise $N+1$ models to generate a starting simplex, for a problem with dimension N;
2. Reflect and perturb the model with the highest objective function value in the simplex, $\mathbf{m}_{\text{high}}$, to get $\mathbf{m}_{\text{reflPert}}$; cf. figure 3.2. Check with the two SA criteria (see section 3.1.1) if $\mathbf{m}_{\text{reflPert}}$ is to replace $\mathbf{m}_{\text{high}}$ in the simplex;
3. If the objective function value of the reflected and perturbed model, $\mathbf{m}_{\text{reflPert}}$, fulfills:
    (a) $f(\mathbf{m}_{\text{reflPert}}) \leq f(\mathbf{m}_{\text{low}})$ go to 4;
    (b) $f(\mathbf{m}_{\text{reflPert}}) \geq f(\mathbf{m}_{2^{\text{nd}}\text{high}})$ go to 5.
    (c) Else, go to 6;
4. Expand and perturb $\mathbf{m}_{\text{reflPert}}$, which must have been included in the simplex in this case, to get $\mathbf{m}_{\text{expPert}}$; cf. figure 3.1(c). Check with the first SA criterion (see section 3.1.1) if $\mathbf{m}_{\text{expPert}}$ is to replace $\mathbf{m}_{\text{reflPert}}$ in the simplex. Go to 6;
5. Contract and perturb one of $\mathbf{m}_{\text{high}}$ and $\mathbf{m}_{\text{reflPert}}$ that is present in the simplex to get $\mathbf{m}_{\text{conPert}}$; cf. figure 3.1(d) and (e). Check with the two SA criteria (see section 3.1.1) if $\mathbf{m}_{\text{conPert}}$ is to replace the model in the simplex which was contracted. Go to 6;

Table 3.4: Second part of a summary of the ASSA algorithm. Continued from table 3.3.

6. If at least one new model replacement has taken place in the simplex, the number of accepted models is increased by one. If enough models have been accepted at the present temperature, go to 7, else go to 2.
7. Reduce temperature. Unless the required number of temperature steps has been taken, go to 2.

3.3 The Contractions

Both types of contractions (seen in figure 3.1(d) and (e)) are performed in the ASSA code, although this is not emphasised in the papers [3] and [6]. In the papers, the impression given is that only the first kind of contraction is made. The different circumstances under which the different contractions are made can be read from tables 3.3 and 3.4. Apparently, $f(\mathbf{m}_{\text{low}}) < f(\mathbf{m}_{\text{reflPert}})$ as well as $f(\mathbf{m}_{2^{\text{nd}}\text{high}}) \leq f(\mathbf{m}_{\text{reflPert}})$ must be fulfilled. The type of contraction is determined by whether or not $\mathbf{m}_{\text{reflPert}}$ was included in the simplex.

3.4 Enhancements

When testing the algorithm on several synthetic test problems, a few problems arose. The program got caught in infinite loops and it could not terminate. The solutions presented below may not be the best but they eliminate the risk of infinite loops.

3.4.1 Mirroring

In the process of generating a new point, there is a chance that the simplex is situated such that the reflections (see figure 3.1(b),(c) and (e)) create a new model outside the search domain (see figure 3.3). A perturbation is added to the new model. If the perturbation is large (i.e. the temperature is high), there is a chance of moving the model inside the search domain. If the perturbation does not succeed in getting the model inside, the algorithm tries to perturb it once again and continues so until it has succeeded. If the temperature is low, the probability that the new model is perturbed into the search domain is very small, and the risk of getting effectively caught in an infinite loop is considerable.

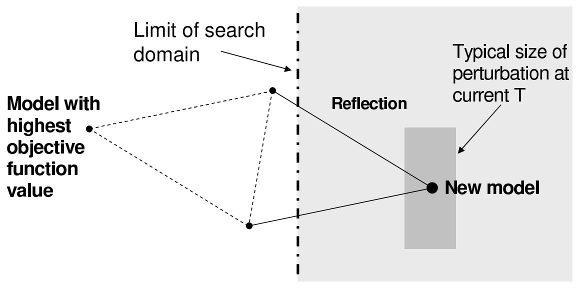

Limit of search domain
Typical size of perturbation at current T
Model with highest objective function value
Reflection
New model

Figure 3.3: A case in which ASSA can get trapped in an infinite loop. A simplex is reflecting the model with highest objective function value outside the search domain. The temperature is very low and the new model is therefore not perturbed enough to get inside the search domain again. The algorithm tries over and over again to perturb the reflected model, but the chance of a perturbation large enough is so small that the algorithm effectively gets caught in an infinite loop.

To avoid that perturbations are tried too many times (or even that the algorithm gets trapped in an infinite loop), a control parameter (`mirrorIndex`) has been introduced to trigger a mirroring of the latest calculated model (after it has been perturbed) into the search domain. The mirroring starts after `mirrorIndex` of perturbations have been tried and discarded.

3.4.2 Multiple Contraction

Another problem with infinite loops can for example take place when having a hill (models with high objective function values) between the best models included in the simplex (see figure 3.4).

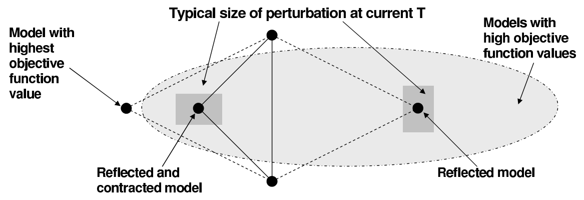

Typical size of perturbation at current T
Model with highest objective function value
Models with high objective function values
Reflected and contracted model
Reflected model

Figure 3.4: Another case in which ASSA can get trapped in an infinite loop. A simplex is reflecting the model with highest objective function value, but there is a hill with high objective function values resulting in another model with high objective function value. The temperature is low and the new model is therefore only slightly perturbed, and it is not accepted by the second SA criterion. The algorithm tries over and over again, but with almost the same result, and it can not escape from the situation.

The algorithm reflects the model as illustrated in figure 3.4. The reflected model is perturbed, but for certain hills, it is probable that the objective function value of $\mathbf{m}_{\text{pert}}$ is higher than that of $\mathbf{m}_{\text{high}}$. The reflected model is not accepted by the first SA criterion. ASSA tries to reflect, contract and perturb over and over again without getting a new model with a lower objective function value such that it would be accepted by the first SA criterion (see page 13). When the temperature is low, the perturbations are small, and in practice, the new models may not be accepted by the second criterion either. Therefore, the algorithm may try over and over again without getting off the hill and it may get effectively trapped in an infinite loop.

This has been taken care of by including a multiple contraction on occasions as described above. The multiple contraction is triggered if the objective function value for $\mathbf{m}_{\text{pert}}$ is greater than or equal to the highest objective function value in the simplex and if the number of times the simplex has been left unchanged is greater than (`multContrIndex`).

3.4.3 Control Parameters

A short description of the control parameters are given here along with suggested values (after [6], [3]). The values suggested within the parentheses may not give the best results, but then have shown to work reasonably well for most problems.

- Initial temperature T_0 or `t0fakt`, where `t0fakt` is used to calculate an initial temperature when there is no knowledge about the problem: T_0=`t0fakt`$\cdot\Delta f_{max}$, where Δf_{max} is the maximum f difference in the initial simplex. (It is chosen such that about 90% of the models are accepted at the start)
- Cooling rate (β = 0.95-0.995 for easy to very hard problems, but it can be lower for very easy problems)
- Number of temperature steps (`nTemp` = 1500)
- Number of accepted perturbations required at each temperature value (`nPert` = 5)
- Factor which controls the temperature dependence of the Cauchy distribution (`nPertfactor` = 4)
- Number of previously successful perturbations to average over when the width of the Cauchy distribution is determined (S = 20)
- Factor for the width of the Cauchy distribution (s = 3.0)
- Number of times to randomly pick a new model before starting to mirror it into the search domain (`mirrorIndex` = 10) (an additional parameter by the author, see section 3.4.1)
- Number of times to try a new model before trying a multiple contraction (`multContrIndex` = 2000) (an additional parameter by the author, see section 3.4.2)

The temperature is on average decreased with the factor $\beta^{\frac{1}{\texttt{nPert}}}$ per accepted new model.

The width of the Cauchy distribution is calculated as s times the running average of the S last successful perturbations times a factor. This factor is decreased with the factor β when `nPertFactor` $\cdot$ `nPert` DHS iterations have been made with a perturbation after each step. One DHS iteration includes step 2 through step 6 in table 3.2.

3.5 Results

For ASSA, variations have been made within two groups of control parameters. One group is constituted by β, `nPert` and `nPertFactor` and the other group by S and s. A reference phase has been chosen for each test example, which is the same for both groups of control parameters. In each figure, the reference phase is the first phase (the blue dotted and dashed curve). The different control parameters have been varied with the reference phase as a starting point. In general, a phase giving a good result quickly was chosen as the reference phase. A good result is used for a phase with high success fraction and a fast ascent.

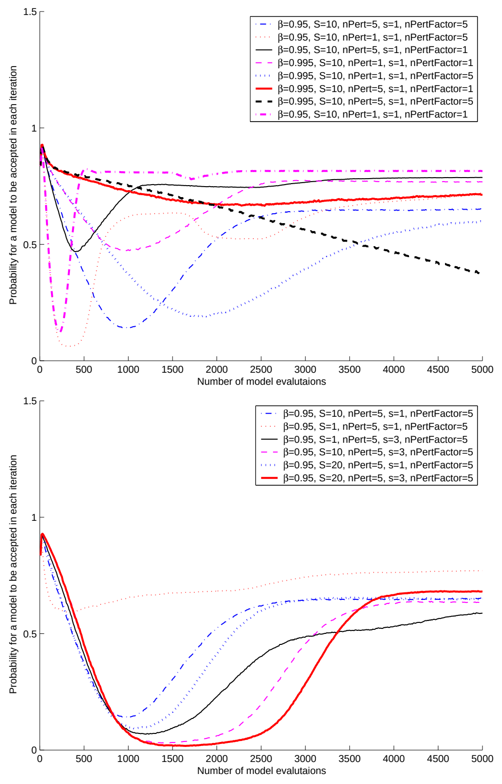


Figure 3.5: The statistics results for the test problem Mexican hat. The upper panel shows group 1 and the lower group 2. Both panels show the estimated probability for a model to be accepted after a certain number of model evaluations. The reference phase is in this case (`t0fakt`, β, S, `nPert`, s, `nPertFactor`) = (20.0, 0.95, 10, 5, 1.0, 5).

A reference phase is useful, since the number of phases quickly becomes cumbersome. Only parameter variations within each group have been tested here.

The parameters T_0 or `t0fakt` are kept constant for all phases for each test problem. The parameters were chosen such that the probability for a model to be accepted should be approximately 90% at the start (see figure 3.5).

For each test problem, the estimated success probabilities and the mean deviations from the optimum are shown in separate figures for the different groups of control parameters.

In the test problems, the values of the control parameters `mirrorIndex` and `multContrIndex` are 10 and 2000, respectively, except for Shubert for which `multContrIndex` = 50.

Mexican Hat

The phase (`t0fakt`, β, S, `nPert`, s, `nPertFactor`) = (20.0, 0.95, 10, 5, 1.0, 5) was chosen as the reference phase for the test problem Mexican Hat (see figures 3.6 and 3.7). The reference phase is in this case also the best phase with a success fraction of 100% and a fast ascent.

Group 1
The phase (20.0, 0.95, 10, 1, 1.0, 1) (see figure 3.6) is the fastest phase, but it often gets stuck in local minima. This phase reaches about 85%. (20.0, 0.995, 10, 1, 1.0, 1) is the phase coming closest to performing as well as the reference phase. It is only slightly slower and reaches the success fraction 100% after about 2750 evaluations compared to about 2000 for the reference phase.

Group 2
For the second group, it can be seen in figure 3.7 that the poorest result (success fraction 65%) is obtained by the phase (20.0, 0.95, 1, 5, 1.0, 5), and the result closest to the reference phase is obtained by the phase (20.0, 0.95, 20, 5, 1.0, 5) which needed only about 250 more evaluations than the reference phase.

Fallat-Dosso

The reference phase for the Fallat-Dosso test problem is (`t0fakt`, β, S, `nPert`, s, `nPertFactor`) = (10.0, 0.975, 15, 10, 1.0, 10) (see figures 3.8 and 3.9). None of the phases in either group reaches success fraction 100%, but the reference phase reaches about 90% which is the highest maximum of the phases.

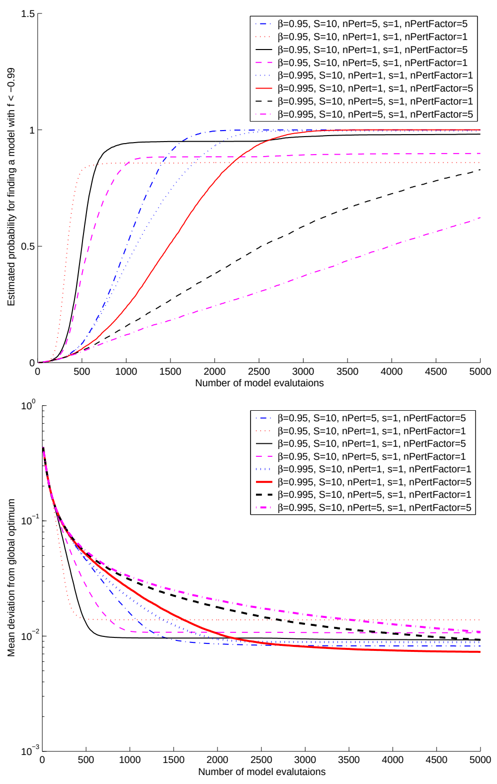


Figure 3.6: The statistics results for the test problem Mexican hat, group 1. The reference phase is in this case (`t0fakt`, β, S, `nPert`, s, `nPertFactor`) = (20.0, 0.95, 10, 5, 1.0, 5). The upper panel shows the estimated probability for finding an objective function value below the limit. The lower panel shows the mean deviation from the global optimum.

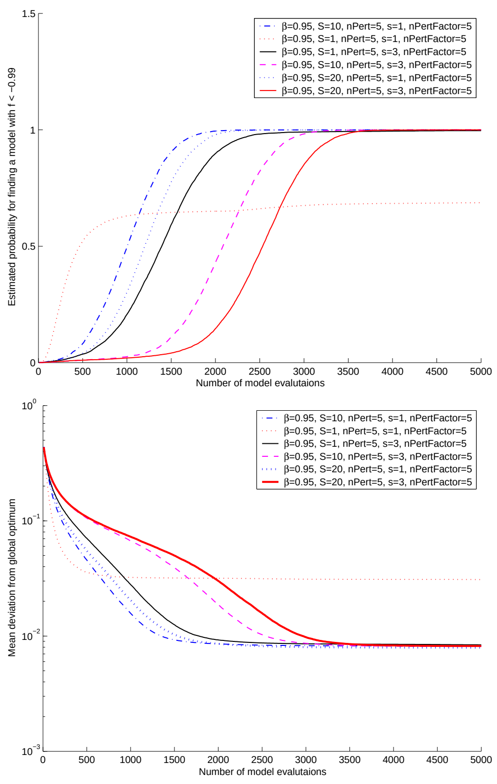


Figure 3.7: The statistics results for the Mexican hat test problem, group 2. The reference phase is (`t0fakt`, β, S, `nPert`, s, `nPertFactor`) = (20.0, 0.95, 10, 5, 1.0, 5). The upper panel shows the estimated probability for finding an objective function value below the limit. The lower panel shows the mean deviation from the global optimum.

Group 1
Of all the phases which can be seen in figure 3.8, the three worst performing phases also have slow ascents. The phase (10.0, 0.995, 15, 1, 1.0, 1) reaches only about 20% and the phases closest to the reference phase are (10.0, 0.975, 15, 5, 1.0, 10), (10.0, 0.975, 15, 10, 1.0, 5) and (10.0, 0.995, 15, 1, 1.0, 10).

Group 2
The phase (10.0, 0.975, 30, 10, 1.0, 10) gives a result almost identical to that of the reference phase. The phase (10.0, 0.975, 1, 10, 1.0, 10) has an early take-off, but it reaches only about 20%.

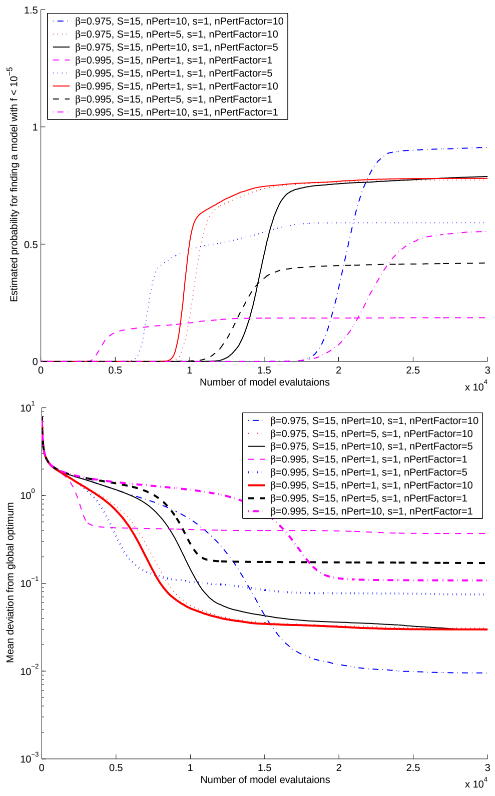


Figure 3.8: The Fallat-Dosso problem, group 1. Here, the phase (`t0fakt`, β, S, `nPert`, s, `nPertFactor`) = (10.0, 0.975, 15, 10, 1.0, 10) was chosen as the reference phase. The upper panel shows the estimated probability for finding an objective function value below the limit. The lower panel shows the mean deviation from the global optimum.

Easom

The reference phase for Easom is (T_0, β, S, `nPert`, s, `nPertFactor`) = (10.0, 0.8, 20, 1, 1.0, 5). Note that in this case, the parameter `t0fakt` is exchanged for T_0.

Group 1
All phases in this group of control parameters perform well for this test problem. The phase (10.0, 0.8, 20, 1, 1.0, 1) is quite close to reaching 100% success fraction with 98% and it is also slightly faster than the reference phase. The phases (10.0, 0.8, 20, 5, 1.0, 1) and (10.0, 0.95, 20, 1, 1.0, 1) give similar results, and they are the ones closest to perform as well as the reference phase.

Group 2
The results for the phases in this group of parameters are more varying than for group 1. The phase (10.0, 0.8, 1, 1, 1.0, 5) is fast but reaches only about 70% while the phase (10.0, 0.8, 20, 1, 3.0, 5) is just a little slower than the reference phase.

Goldstein-Price

The Goldstein-Price reference phase is

(`nsamplei`, β, S, `nPert`, s, `nPertFactor`) = (20.0, 0.95, 1, 1, 3.0, 1).

Group 1
The phase (20.0, 0.995, 1, 5, 3.0, 1) does not succeed at all before 5000 evaluations. The phase (20.0, 0.95, 1, 1, 3.0, 5) has the same behaviour as the reference phase, with the same slope for its ascent. The difference is that the reference phase reaches 100% after 600 evaluations, while (20.0, 0.95, 1, 1, 3.0, 5) needs about 1100 evaluations.

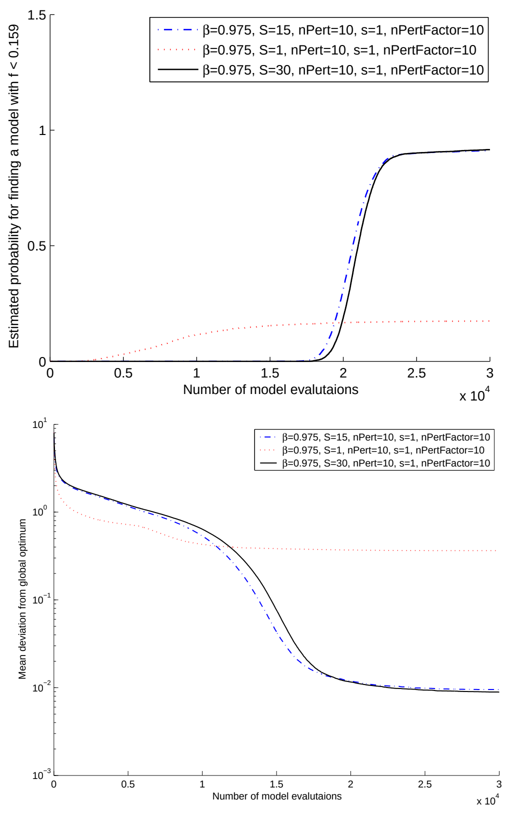


Figure 3.9: The test problem Fallat-Dosso, group 2. The reference phase is (`t0fakt`, β, S, `nPert`, s, `nPertFactor`) = (10.0, 0.975, 15, 10, 1.0, 10). The upper panel shows the estimated probability for finding an objective function value below the limit. The lower panel shows the mean deviation from the global optimum.

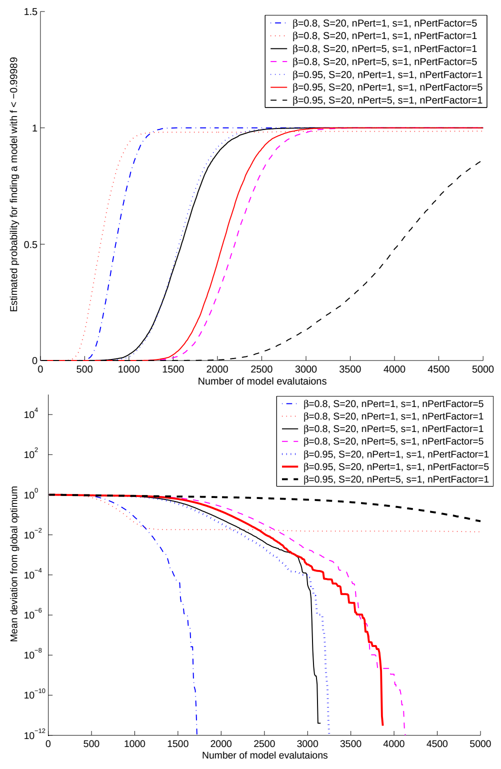


Figure 3.10: The Easom test problem, group 1. The reference phase is (T_0, β, S, `nPert`, s, `nPertFactor`) = (10.0, 0.8, 20, 1, 1.0, 5). The upper panel shows the estimated probability for finding an objective function value below the limit. The lower panel shows the mean deviation from the global optimum.

Group 2
All but one phase in this group have about the same behaviour. The phase which deviates is (20.0, 0.95, 1, 1, 1.0, 1). This phase has an earlier take-off. Among the other phases, (20.0, 0.95, 10, 1, 1.0, 1) is the one coming closest to the reference phase.

Shubert

The reference phase for the test problem Shubert is (`nsamplei`, β, S, `nPert`, s, `nPertFactor`) = (30.0, 0.9, 30, 1, 3.0, 4).

Note the difference in number of evaluations in the different groups.

Group 1
For this test problem and these control parameters, four subgroups appear with different ascent slopes. The phase (30.0, 0.9, 30, 1, 3.0, 2) and the reference phase have the fastest ascent and the earliest take-off, although (30.0, 0.9, 30, 1, 3.0, 2) does reach about 97%. The other subgroups have much slower ascents, and two phases have just left the bottom at 10000 evaluations.

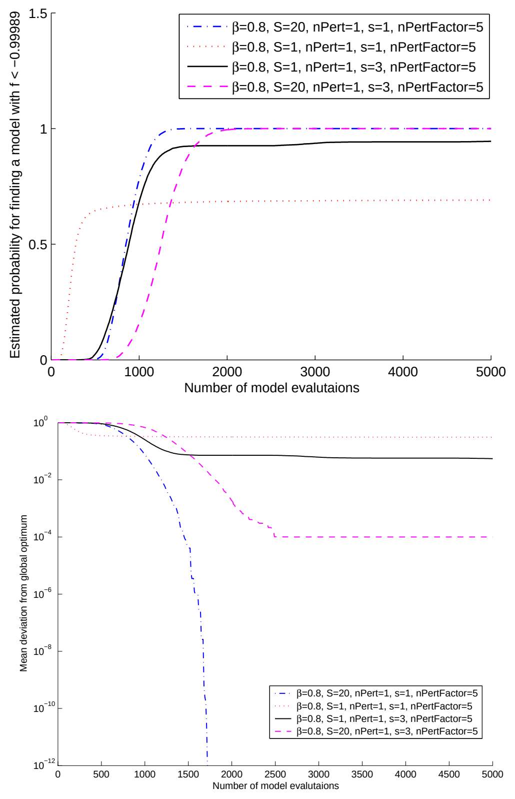


Figure 3.11: The results for the test problem Easom, group 2. The reference phase was chosen to be (T_0, β, S, `nPert`, s, `nPertFactor`) = (10.0, 0.8, 20, 1, 1.0, 5). The upper panel shows the estimated probability for finding an objective function value below the limit. The lower panel shows the mean deviation from the global optimum.

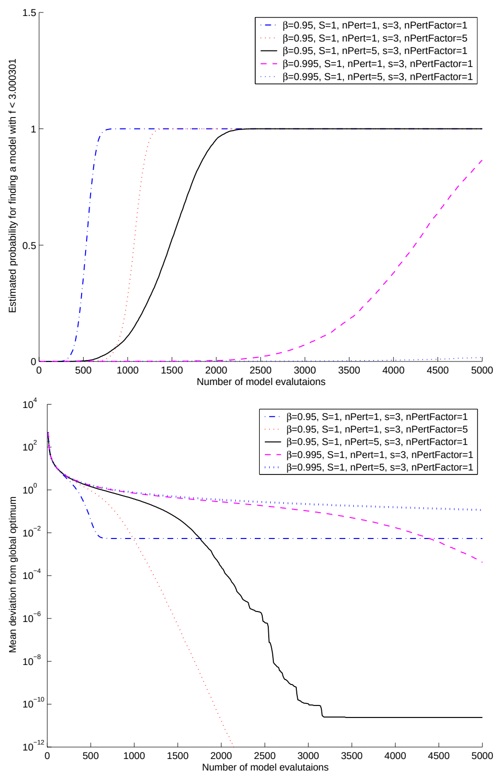


Figure 3.12: The results for the Goldstein-Price problem, group 1. The reference phase is the phase (`nsamplei`, β, S, `nPert`, s, `nPertFactor`) = (20.0, 0.95, 1, 1, 3.0, 1). The upper panel shows the estimated probability for finding an objective function value below the limit. The lower panel shows the mean deviation from the global optimum.

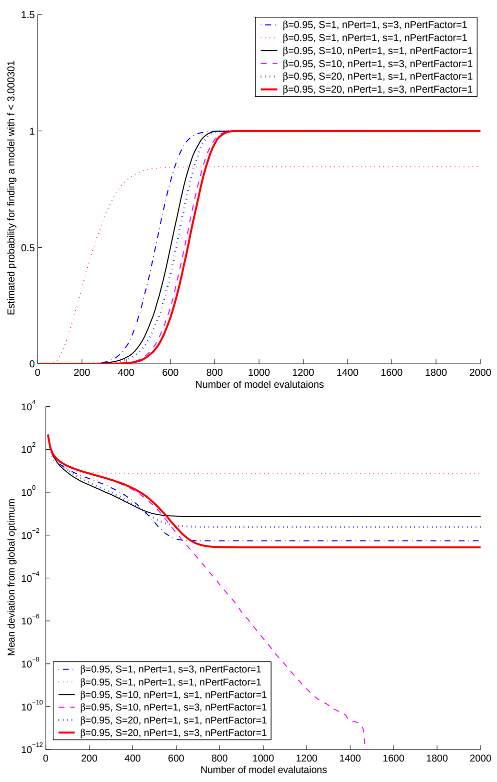


Figure 3.13: The statistics result for the test problem Goldstein-Price, group 2. The reference phase for this problem is (`nsamplei`, β, S, `nPert`, s, `nPertFactor`) = (20.0, 0.95, 1, 1, 3.0, 1). The upper panel shows the estimated probability for finding an objective function value below the limit. The lower panel shows the mean deviation from the global optimum.

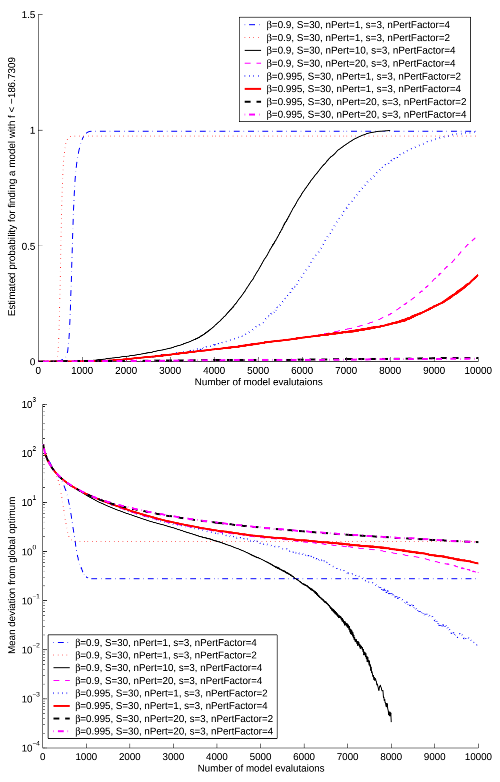


Figure 3.14: The statistics result for the problem Shubert, group 1. The reference phase is (`nsamplei`, β, S, `nPert`, s, `nPertFactor`) = (30.0, 0.9, 30, 1, 3.0, 4). The upper panel shows the estimated probability for finding an objective function value below the limit. The lower panel shows the mean deviation from the global optimum.

Group 2

All phases in this group of control parameters have almost the same ascent slope. The differences concern the take-off of the curves and whether or not 100% is really reached. The phase (30.0, 0.9, 10, 1, 1.0, 4) is fastest, and it has reached 97% after less than 800 evaluations. The reference phase has the latest take-off, and it reaches 100% after 1200 evaluations.

`mirrorIndex`

The `mirrorIndex` control parameter is often quite low (10) in the examples, but can be chosen both higher and lower, depending on what suits the specific problem. As already pointed out, the main advantage with mirroring is that the threat of getting trapped in certain long loops is eliminated.

Tests have been made to compare the following two cases with the original one: 1. only mirroring (instead of continued perturbations) 2. `mirrorIndex` = 10 (see section 3.4.1). The

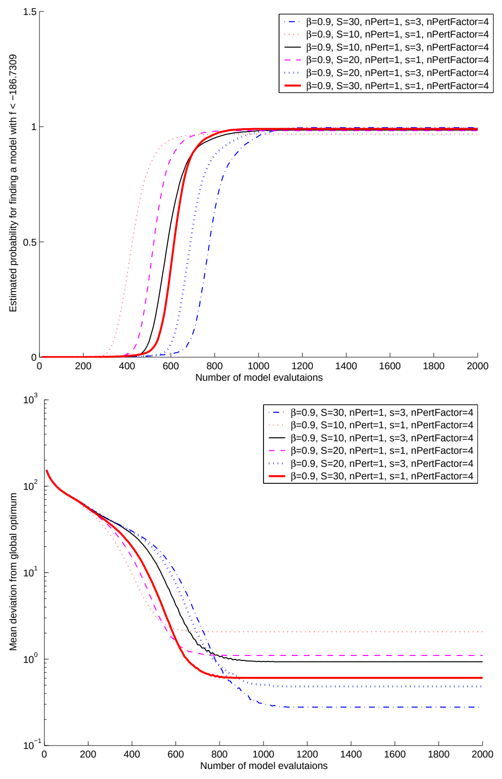


Figure 3.15: The statistics result for the test problem Shubert, group 2. The reference phase is the phase (`nsamplei`, β, S, `nPert`, s, `nPertFactor`) $= (30.0, 0.9, 30, 1, 3.0, 4)$. The upper panel shows the estimated probability for finding an objective function value below the limit. The lower panel shows the mean deviation from the global optimum.

original case included only perturbing. The differences between the original case and the two other cases with mirroring were not bigger than the uncertainty in the probability estimation for these examples, for which the long-loop problem was not critical.

`multContrIndex`

The control parameter `multContrIndex` is usually set to a high number (2000), but for certain kinds of problems ASSA can produce better results if `multContrIndex` is changed. As can be seen in figure 3.16, the value of `multContrIndex` is very important when working with the test problem Shubert. When `multContrIndex` $= 2000$, plateaus appear with length around 2000 evaluations, but when decreasing the value to `multContrIndex` $= 50$ the plateaus can not be seen. Also note that the method gives good results much faster with more multiple contractions.

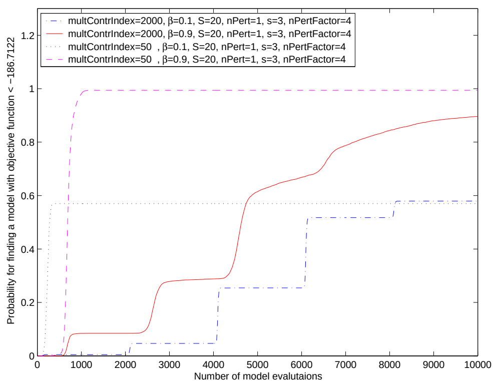


Figure 3.16: Four phases for the test problem Shubert. This test problem causes troubles for the algorithm as can be seen by the plateau shape in two of the curves. This could be avoided by shifting `multContrIndex` to a smaller number (from 2000 to 50 in this case), as can be seen by comparing to the two curves without plateaus. Notice that the results are better or similar with much faster ascents of the curves.

3.6 Discussion

The reason why ASSA does not produce as good values for the Fallat-Dosso problem as the other test problems may be found in the control parameters. If there was more time to adjust the parameters, ASSA could perhaps improve the results.

Group 1 Control Parameters
When the value of the cooling rate, β, is changed, the ascent is changed and/or the take-off of the success fraction curve is changed. This behaviour can be observed in all figures for group 1, when only β is changed. When β is increased the ascent is slower, and when β is decreased the ascent is faster.

The algorithm becomes more greedy[2] in its search when `nPert` or `nPertFactor` are lowered separately (see the control parameter section, page 20). The search seems to go even faster if `nPert` and `nPertFactor` are lowered simultaneously, but it also gets stuck in local minima more often. If β is increased, the ascent is slower, but if `nPert` and/or `nPertFactor` are decreased simultaneously, the affect of the increase in β can be counteracted. This behaviour can be seen in figure 3.6.

The change in `nPertFactor` does not seem to have such a drastic effect as that in `nPert` has. When `nPert` is increased the ascent is much slower, as can be seen in figure 3.14.

Group 2 Control Parameter
For the parameter s (see the control parameter section, page 20), in general, an increase leads to a more greedy search and a decrease leads to a more slow and thorough search (see figure 3.7). When s is decreased, the algorithm becomes more and more local. When $s = 0$, the algorithm is very similar to pure DHS.

In the Fallat-Dosso problem (see figure 3.9), the effect of changing S can be observed. If S is increased, there is almost no effect, but if S is decreased instead, there is a drastic effect: the take-off is earlier, but the success is only approximately 20%. Apparently, a change of S from a small to a moderate value has a larger effect than a change from a moderate to a large value. The interpretation for S is not as simple as that for s. It seems natural, however, that there is a larger difference between, for example, the running averages of the last two and the last ten perturbations than between the running averages of the last ten and the last twenty perturbations.

In figure 3.11 and figure 3.15, the behaviour when both S and s are changed can be observed. If S is decreased and s is increased, the take-off is delayed. If the parameters are changed in opposite directions, they may counteract to some extent.

The large number of control parameters is of course a drawback of the technique. Still, it is manageable because some of the control parameters are seldom changed (as for example, `multContrIndex` and `mirrorIndex`) and because the suggested numbers (see section 3.4.3) often provide good starting points for tuning.

[2]A greedy algorithm considers only what is best for the moment and not in the long run. In most cases, this means that the algorithm always uses the best model yet found as a basis for its creation of new models. Increased greediness usually means faster optimisation at the cost of increased risk of getting trapped in local minima.

Chapter 4

Differential Evolution

Rune Westin

4.1 Background

Differential Evolution (hereafter called DE) was developed by Rainer Storn and Kenneth Price, in the mid nineties [9], [10]. Their idea was to find a simple, fast and reliable global optimisation algorithm.[1]

The basics for DE came out of Kenneth Price's attempts to solve a Chebychev Polynomial fitting problem. Price proposed the method of using differences between models to perturb the model population and a discussion between Price and Storn commenced. Since 1994-1996, many improvements on DE have been made by Storn, Price and an increasing community of researchers working on the algorithm.

DE is much related to *genetic algorithms* and other *evolutionary algorithms* in the sense that it works with a *population* of models evolving towards the global optimum. Hence, is is useful to start with a brief description of genetic algorithms.

4.1.1 Genetic Algorithms

The genetic algorithm is a natural evolution-inspired stochastic search method that operates on a population Pop of models $\mathbf{m}_i$ [2].

$$Pop = \{\mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_{NP}\},$$

where NP is the size of the population. In some cases, it is convenient to refer to a model as a *member* of the population. Each model $\mathbf{m}_i$ has an objective function value[3] $f(\mathbf{m}_i)$ and the purpose of the algorithm is to minimise f by updating the population for a number of *generations*.

In order to generate the next generation, the algorithm combines different members of the population into new models and selects the best models, keeping the population size NP constant.

The models must be stored in a way suitable for computers. Each variable is limited by boundaries based on some prior knowledge of the problem, and the j:th variable can be discretised as a string of n_j bits (n_j is a positive integer). This gives 2^{n_j} possible values between the boundaries, where n_j corresponds to the required resolution for the j:th variable. The variable bit strings can then be put together to form a model string with $\sum_j n_j$ bits

[1] General information and codes for a wide variety of compilers can be found at the Differential Evolution Homepage [11].

[2] In genetic algorithms, a model is often called a *genome* or a *chromosome* and the variables it consists of are called *genes*.

[3] The objective function value is usually referred to as *fitness* in genetic algorithms.

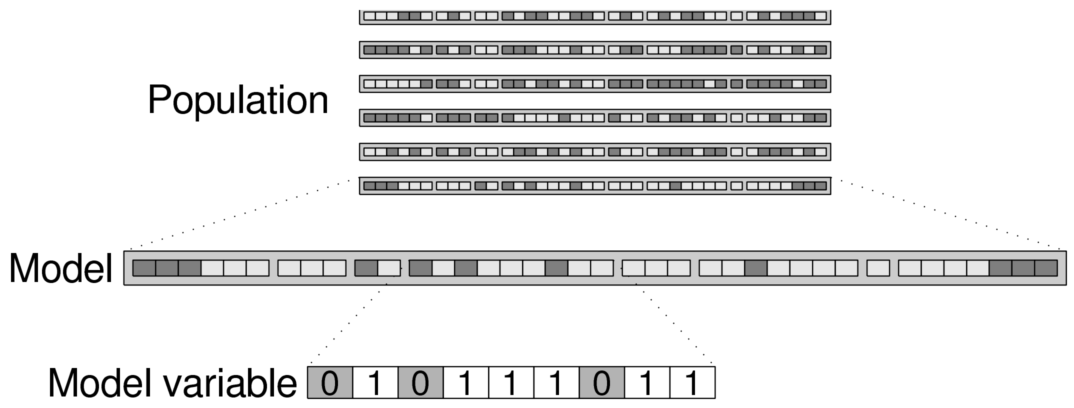


Figure 4.1: Illustration of population, model and variable as strings of bits.

(see Figure 4.1).

The genetic algorithm is outlined in table 4.1 and described more thoroughly in what follows.

Population Initialisation

The genetic algorithm starts with the initialisation of the population. *NP* models are randomly created and the objective function values f for all member models are evaluated.

Offspring Creation

In every generation, a certain number of offspring models are created, each of which has two parent models. The parents are randomly chosen with probabilities based on their objective function values. Models with low f have higher probabilities to be chosen as parents than models with high f.

The new models are created with their characteristics taken from their parents in a process called *crossover*. There are many variations of the genetic algorithm and the crossover process differs between them. In some, each individual bit is taken from parents randomly. In others, whole variables are inherited.

Offspring Mutation

The offspring models are also mutated to obtain more diversity in the population. This is usually done by randomly choosing a few bits to "flip" in the model string [4].

Next Generation

The created offspring models and the members of the previous generation are sorted according to their objective function values. The *NP* models with the lowest f values are kept to the next generation of the population. The remaining models are discarded.

There are variations of this step as well.

Termination Criterion

The algorithm is stopped either when a model with a sufficiently low objective function value has been found or when a predetermined number of generations have passed.

[4]A bit is flipped by changing it from 0 to 1 or from 1 to 0.

Table 4.1: Basic outline of genetic algorithms

1. Initialise a population of *NP* randomly chosen models.
2. Evaluate the objective function value f for each member model.
3. Proceed iteratively until a member with a sufficiently low f has been found, or until a predetermined number of generations have passed:
   (a) Select the members that should mate according to their objective function value rank in the population.
   (b) Create offspring members that inherit characteristics from their parents. Mutate the offspring.
   (c) Calculate f for each new member.
   (d) Remove the members with the highest objective function values.

Table 4.2: Basic outline of Differential Evolution

1. Initialise a population of *NP* randomly chosen models $\mathbf{m}_i$.
2. Calculate the objective function value f_i for each model $\mathbf{m}_i$.
3. Proceed iteratively until a model with a sufficiently low f has been found, or until a predetermined number of generations have passed:
   (a) For each model $\mathbf{m}_i$, randomly select three other member models $\mathbf{m}_{r_1(i)}$, $\mathbf{m}_{r_2(i)}$ and $\mathbf{m}_{r_3(i)}$.
   (b) Create *NP* new models $\mathbf{v}_i = \mathbf{m}_{r_1(i)} + F \cdot (\mathbf{m}_{r_2(i)} - \mathbf{m}_{r_3(i)})$.
   (c) Create *NP* new models $\mathbf{u}_i$, where each variable in $\mathbf{u}_i$ comes from $\mathbf{v}_i$ with probability *CR* and from $\mathbf{m}_i$ with probability $1-CR$.
   (d) Calculate the objective function value for each model $\mathbf{u}_i$. If $f(\mathbf{u}_i) < f(\mathbf{m}_i)$, $\mathbf{m}_i$ is replaced by $\mathbf{u}_i$ for the next generation.

4.2 Description of the DE Algorithm

As stated earlier, DE is related to genetic algorithms (GA) in the sense that it works with a population of *NP* models. There are some differences between DE and GA. The models in GA are bit strings representing some discretisation of the search domain, in close relation to genes and computers. The models in DE are vectors of real numbers, thus giving a continuous representation of the search domain.

In GA, new models inherit bits or bit strings from their parent models. DE, in contrast, uses the difference between model vectors as a basis for its new models. Also, in contrast to GA, there are no mutations in DE.

4.2.1 Outline of DE

I have used a Fortran 77 version of DE based on a Fortran 90 code[5] published on the Differential Evolution home page [11]. The algorithm is outlined in table 4.2 and described in detail here.

[5]The Fortran 90 code was developed by Dr. Feng-Sheng Wang at the department of chemical engineering at National Chung Cheng University in Taiwan.

Population Initialisation

DE starts by creating a population of NP randomly chosen models throughout the search domain. It is recommended to allow as much variation as possible of the model population by not constraining the randomisation.

The objective function value f for each model is evaluated and the model $\mathbf{m}_{\text{low}}$ with the lowest objective function value f_{low} is determined.

The rest of the algorithm is performed in an iterative manner. Each iteration involves the creation of a new generation.

Partner Model Creation

All population member models *mate* to create offspring models, one offspring per population member. In order to mate, each member $\mathbf{m}_i$ must have a mating partner model $\mathbf{v}_i$ which is created for this sole purpose ($i = 1, \ldots, NP$).

For each mating partner $\mathbf{v}_i$, three models – $\mathbf{m}_{r_1(i)}$, $\mathbf{m}_{r_2(i)}$ and $\mathbf{m}_{r_3(i)}$ – are chosen randomly from the population ($r_1(i)$, $r_2(i)$, $r_3(i)$ and i all different). $\mathbf{v}_i$ is calculated as the weighted difference between $\mathbf{m}_{r_2(i)}$ and $\mathbf{m}_{r_3(i)}$ added to $\mathbf{m}_{r_1(i)}$:

$$\mathbf{v}_i = \mathbf{m}_{r_1(i)} + F \cdot (\mathbf{m}_{r_2(i)} - \mathbf{m}_{r_3(i)}) \qquad (4.1)$$

where F is a weighting factor given by the user ($F > 0$). This is repeated NP times with different r_1, r_2 and r_3 each time, giving each model in the population a partner.

The partner creation is illustrated in steps a) through c) in figure 4.2.

Offspring Model Creation (Crossover)

The old population members $\mathbf{m}$ are combined with their partner models $\mathbf{v}$ to create NP offspring models $\mathbf{u}$ in a crossover process. An offspring model $\mathbf{u}_i$ takes variables randomly from its parents $\mathbf{m}_i$ and $\mathbf{v}_i$.

The user given control parameter CR is the probability that a variable in $\mathbf{u}_i$ will be chosen from $\mathbf{v}_i$. Correspondingly, a variable of $\mathbf{u}_i$ will be chosen from $\mathbf{m}_i$ with probability $1 - CR$. This means that in a search domain of N dimensions, there are 2^N possible choices for each $\mathbf{u}_i$, including $\mathbf{v}_i$ and $\mathbf{m}_i$. This is illustrated in step d) in figure 4.2 [6].

Once the NP offspring models $\mathbf{u}$ are created, the parent models $\mathbf{v}$ are discarded.

Selection for the Next Generation

The objective function values for the NP offspring models are evaluated. Each offspring model $\mathbf{u}_i$ is compared to its parent $\mathbf{m}_i$ and the model with the lowest objective function value is selected as $\mathbf{m}_i$ for the next generation.

In order to keep track of the best model found, $f(\mathbf{m}_{\text{low}})$ is compared to the objective function value of the best model in the new generation. If necessary, $\mathbf{m}_{\text{low}}$ is updated.

Termination Criteria

There are two termination criteria.

1. A model with a sufficiently low objective function value has been found.
2. A predetermined number of generations have passed.

If one of these criteria is fulfilled, the algorithm stops. The best model $\mathbf{m}_{\text{low}}$ and its objective function value f_{low} are presented as the proposed minimum.

4.2.2 Algorithm Variations

Storn and Price have proposed some variations of the algorithm concerning the creation

[6]The original DE by Storn and Price uses a slightly different scheme for the crossover described in [9] and [10].

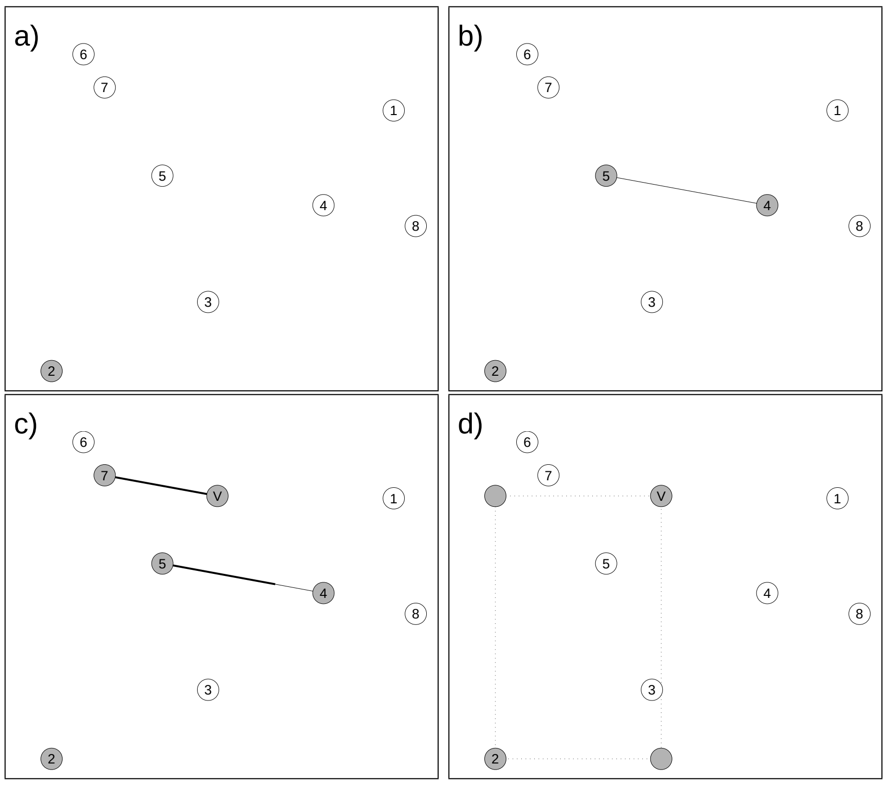


Figure 4.2: Illustration of the partner generation and crossover process. For each model $\mathbf{m}_i$, a partner model $\mathbf{v}_i = \mathbf{m}_{r_1(i)} + F \cdot (\mathbf{m}_{r_2(i)} - \mathbf{m}_{r_3(i)})$ is generated. In this example $i = 2$, $r_1(i) = 7$, $r_2(i) = 4$ and $r_3(i) = 5$. a) The population with $\mathbf{m}_2$ highlighted. b) $\mathbf{m}_5$ and $\mathbf{m}_4$ highlighted with the vector $\mathbf{m}_4 - \mathbf{m}_5$ in between. c) $\mathbf{v}_2 = \mathbf{m}_7 + F \cdot (\mathbf{m}_4 - \mathbf{m}_5)$. d) The four possible choices for the new model $\mathbf{u}_2$ after the crossover process where each variable of $\mathbf{u}_2$ is taken from $\mathbf{v}_2$ with probability CR and from $\mathbf{m}_2$ with probability $1 - CR$.

of partner models. The scheme described in section 4.2.1, equation (4.1), is called DE1 or DE/rand/1.

DE2

Another scheme is called DE2 and includes the best model of the population in the creation of partner models [9]:

$$\mathbf{v}_i = \mathbf{m}_i + \lambda \cdot (\mathbf{m}_{\text{low}} - \mathbf{m}_i) + F \cdot (\mathbf{m}_{r_1(i)} - \mathbf{m}_{r_2(i)}).$$

Here, λ is introduced as a control parameter used to enhance the greediness of the scheme by making use of the best model yet found. This increases the speed of the algorithm but also makes it more prone to get trapped in local minima.

DE/best/2

The scheme called DE/best/2 is also based on the best model of the population, which increases the greediness. Here, a fourth random model $\mathbf{m}_{r_4(i)}$ is included in the generation of $\mathbf{v}_i$:

$$\mathbf{v}_i = \mathbf{m}_{\text{low}} + F \cdot (\mathbf{m}_{r_1(i)} + \mathbf{m}_{r_2(i)} - \mathbf{m}_{r_3(i)} - \mathbf{m}_{r_4(i)})$$

Other Schemes

There are a few more schemes in the Fortran 90 code I based my implementation on:

$$\mathbf{v}_i = \mathbf{m}_{\text{low}} + F \cdot (\mathbf{m}_{r_1(i)} - \mathbf{m}_{r_2(i)})$$

$$\mathbf{v}_i = \mathbf{m}_i + F \cdot (\mathbf{m}_{\text{low}} - \mathbf{m}_i + \mathbf{m}_{r_1(i)} - \mathbf{m}_{r_2(i)})$$

$$\mathbf{v}_i = \mathbf{m}_{r_5(i)} + F \cdot (\mathbf{m}_{r_1(i)} - \mathbf{m}_{r_2(i)} + \mathbf{m}_{r_3(i)} - \mathbf{m}_{r_4(i)})$$

$$\mathbf{v}_i = \mathbf{m}_i + F_{CR} \cdot (\mathbf{m}_{\text{low}} - \mathbf{m}_i) + F \cdot (\mathbf{m}_{r_1(i)} - \mathbf{m}_{r_2(i)}).$$

The last scheme includes a linear crossover combination of $\mathbf{m}_i$ and $\mathbf{m}_{\text{low}}$. F_{CR} is a random number $0 \leq F_{CR} \leq 1$.

Of all the scheme variations, I found DE/rand/1, described in section 4.2.1, to give the best overall results in my tests. All results described henceforth are obtained with DE/rand/1.

4.2.3 Control Parameters

The user has five control parameters to set. Two of them concern the stopping criteria: the objective function value for which the algorithm stops and the maximum number of generations.

This leaves three control parameters that actually control the behaviour of DE:

- The population size NP. NP is a positive integer and is usually set to five to ten times the dimension N of the search problem.
- The weighting factor F is a positive real number, usually smaller than 1.
- The crossover factor CR. CR is a probability and is therefore a real number between 0 and 1.

4.3 Results

To obtain knowledge of the effect of different choices of control parameter values, DE was run on the five synthetic test functions described in chapter 2. I chose two different values for each of the three control parameters and tested the eight possible combinations (i.e. phases). The number of runs in each phase (n_{run}) was 10000.

The choices for the population size NP were $NP = 5N$ and $NP = 10N$, where N is the dimension of the test function. The weighting factor was set to $F = 0.5$ and $F = 0.9$. The crossover factor was set to $CR = 0.5$ and $CR = 0.9$.

Mexican Hat

The probability curves for the different phases seen in figure 4.3 can be divided into four groups with varying rise speed and reliability. The fasted phase has a low population size, a high crossover factor and a low weighting factor. The slowest phase has the opposites of these characteristics.

The phases in between seem to group themselves according to how many control parameters have values similar to the slowest or the fastest phase.

Generally in Mexican hat, slower rise speed means higher reliability.

Fallat-Dosso

Figure 4.4 shows the results of the DE optimisations on Fallat-Dosso. The fastest phase is the same as in the Mexican hat optimisation. The fastest phase with a high reliability has a low population size, a low crossover factor and a low weighting factor. Two of the phases have not risen above $P_{\text{est}} = 0.01$ within the given number of evaluations.

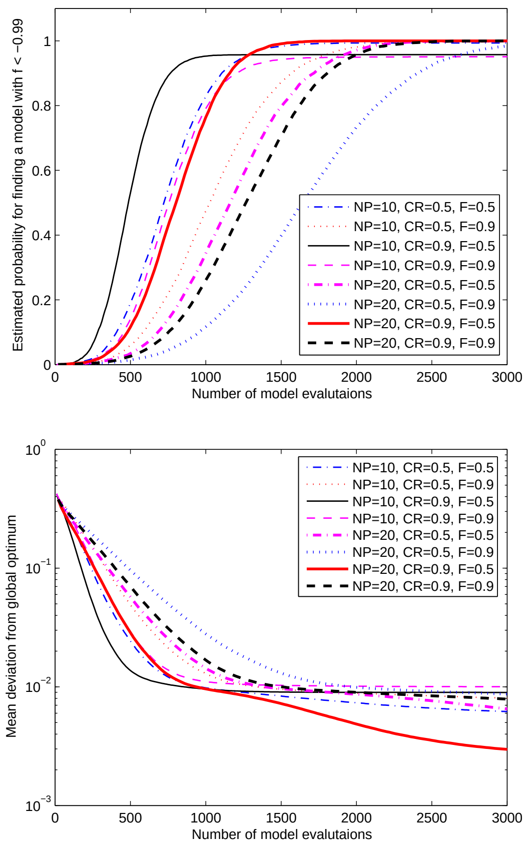


Figure 4.3: DE optimisation of Mexican hat. *Top*: The estimated success probabilities. *Bottom*: The mean deviations from the global optimum.

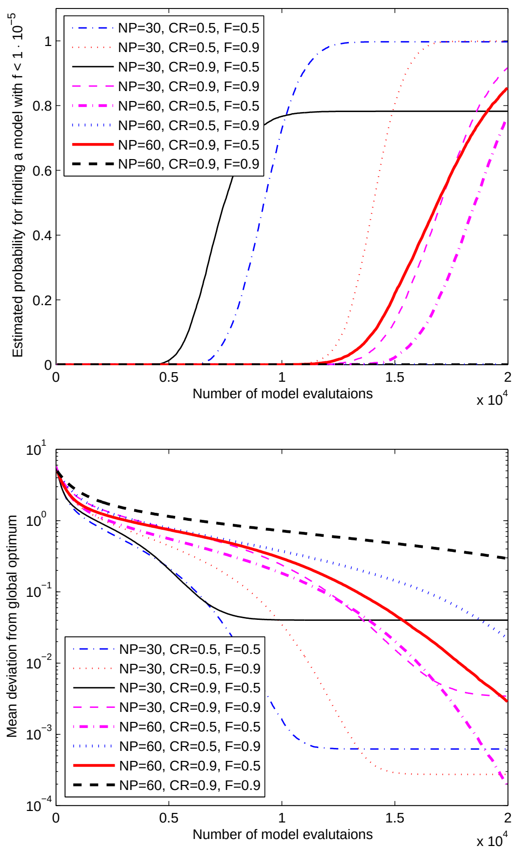


Figure 4.4: DE optimisation of Fallat-Dosso. *Top*: The estimated success probabilities. *Bottom*: The mean deviations from the global optimum.

Easom

The results from the DE optimisation of Easom are seen in figure 4.5. The fastest phase is the same as for the previous test functions. Here, the fastest phase with a high reliability has a high population size, a high crossover factor and a low weighting factor.

Goldstein-Price

Figure 4.6 shows the results from the DE optimisation of Goldstein-Price. Again, the fastest phase has a low population size, a high crossover factor and a low weighting factor. There are two fast phases with a high reliability. Both have high crossover factors but one has a high population size and the other has a high weighting factor. The former of these phases is more reliable while the latter phase is faster.

Shubert

As seen in the DE results from Shubert in figure 4.7, the fastest phase has a low population size, a high crossover factor and a low weighting factor. Again, there are two phases that combine speed and reliability. Both have

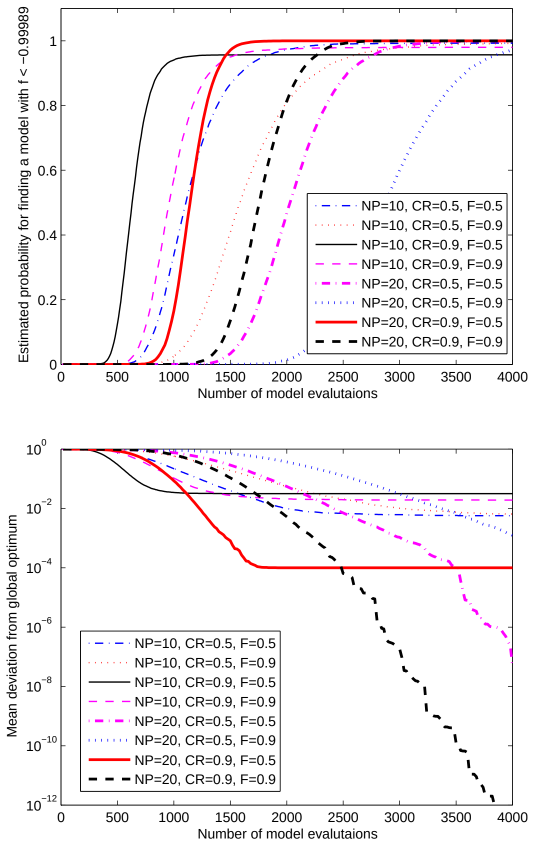


Figure 4.5: DE optimisation of Easom. *Top*: The estimated success probabilities. *Bottom*: The mean deviations from the global optimum.

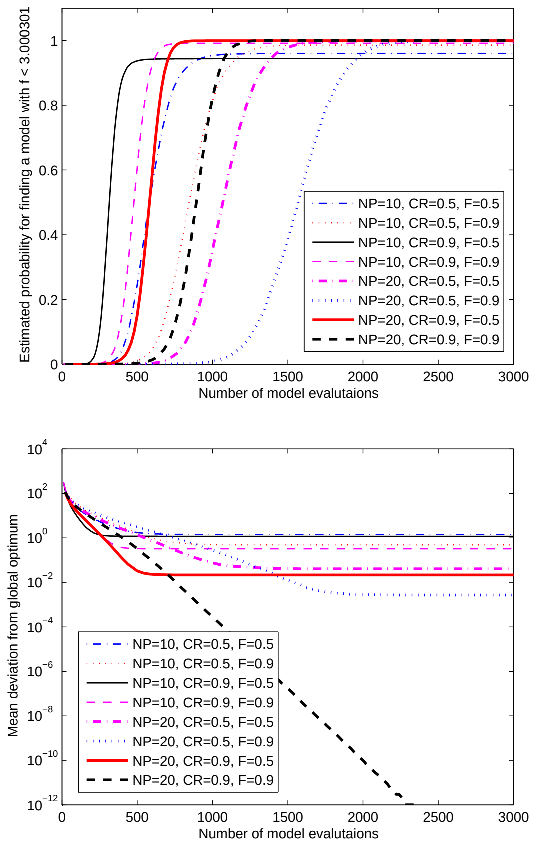


Figure 4.6: DE optimisation of Goldstein-Price. *Top*: The estimated success probabilities. *Bottom*: The mean deviations from the global optimum.

high population sizes and low weighting factors. The phase with a high crossover factor is faster and the phase with a low crossover factor is more reliable.

4.4 Discussion

The results show that DE is a reliable, fast and versatile optimisation algorithm. The estimated probability reached 100 % for several phases in all test functions. The algorithm is easy to understand and convenient to program. Also, since DE creates a new population of independent models in each generation, it is well suited for a parallel computing implementation.

Another advantage of DE is that there is no problem with varying magnitudes in the ranges of the different variables in the search domain.

For example, consider a two-dimensional objective function where the range of the first variable x_1 is $0 \leq x_1 \leq 0.1$ and the range of the second variable x_2 is $0 \leq x_2 \leq 10^5$. Some optimisation techniques might have to use scaling factors to be able to search the whole range of x_2 while staying between the boundaries of x_1. Since DE bases its new models on the differ-

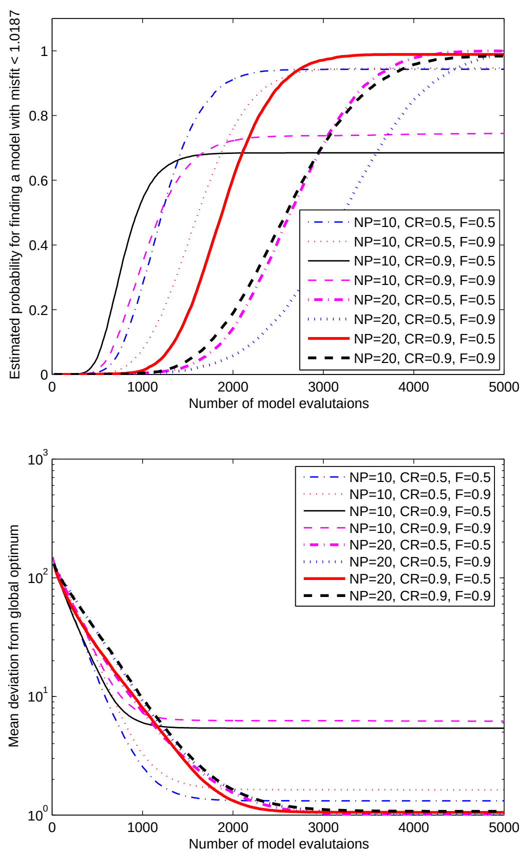


Figure 4.7: DE optimisation of Shubert. *Top*: The estimated success probabilities. *Bottom*: The mean deviations from the global optimum.

ence vectors between old models, no extra care has to be taken to explore the whole search domain.

4.4.1 Choices of Control Parameter Values

Common for all of the DE optimisations in the tests is that there is a trade-off between speed and reliability for the different phases. It is mostly the case that the values of each of the three control parameters can be set to make the optimisation either fast or reliable.

Population Size

It can generally be said that a higher population size means a slower but more reliable optimisation.

On the DE home page [11], there is a suggestion of which control parameter values to start with when facing a new optimisation problem. The recommendation is to chose $NP = 10N$, where N is the number of dimensions in the objective function.

I found that $NP = 5N$, or a slightly larger population size, was enough for most test problems. In an applied problem, however, the global optimum is not known and the extra reliability of a larger population might be desirable despite the extra evaluations needed.

Crossover Factor

According to the DE home page [11], the choice of CR has less influence on performance than the other control parameters and should be used for fine tuning. A suggestion could be to set CR to 0.9, and lower it if the results are not satisfactory.

A higher CR usually means a faster but less reliable optimisation.

Weighting Factor

Concerning the weighting factor, a higher F usually means a slower but more reliable optimisation. The recommendation in [11] is to start with $F = 0.8$ and adapt from there.

4.4.2 Keeping Models within the Search Domain

It is important to check if the partner model $\mathbf{v}$ is created outside the search domain. A model outside the search domain must be moved inside, preferably by mirroring as described below.

If $v_j > x_{j,\max}$, where v_j is the j:th element of $\mathbf{v}$ and $x_{j,\max}$ is the upper boundary for the j:th objective function variable x_j, then $\mathbf{v}$ is mirrored by setting

$$v_{j,\text{mirrored}} = x_{j,\max} - (v_j - x_{j,\max}). \tag{4.2}$$

If $v_j < x_{j,\min}$, where $x_{j,\min}$ is the lower boundary for the j:th objective function variable x_j, then $\mathbf{v}$ is mirrored by setting

$$v_{j,\text{mirrored}} = x_{j,\min} + (x_{j,\min} - v_j). \tag{4.3}$$

It may be necessary to perform these mirroring steps repeatedly.

Another method used in the Fortran 90 code I based my implementation on was to simply change one or more variables of the model to move it to the closest possible place within the search domain. This place is always on a boundary of the domain ($v_{j,\text{moved}} = x_{j,\max}$ or $v_{j,\text{moved}} = x_{j,\min}$), resulting in a gathering of the population at the boundaries.

I found the optimisation results to be better when using the mirroring method compared to the other method, possibly because of the more diverse distribution of the population obtained by mirroring.

Chapter 5

Neighbourhood Algorithm

Hanna Gothäll

5.1 Background

The Neighbourhood Algorithm (NA) was introduced by Sambridge in 1999 [12] and was tested by Resovsky and Trampert in 2002 [13]. A basic question asked by Sambridge was: 'How can a search for new models be best guided by all previous models for which the forward problem has been solved (and hence the data-objective function value evaluated)?' [12]. This question led to the algorithm as it is described in the next section.

The algorithm was originally designed for inverse seismology problems. It has already been prepared for parallel processing.

5.2 Description of the Algorithm

The algorithm is conceptually simple with only two control parameters and with the use of the rank of a data fit criterion instead of a numerical value. Hence, scaling of the objective function values are immaterial.

NA consists of a number of iterations. In the zeroth iteration, n_s models are chosen randomly and the Voronoi cells around these models are calculated. The Voronoi cells are the nearest neighbourhood regions to the models around which they are created. The intersection between two cells are at half the distance between the models as shown in figure 5.1. This is a unique way of dividing the multi-dimensional parameter space into smaller regions.

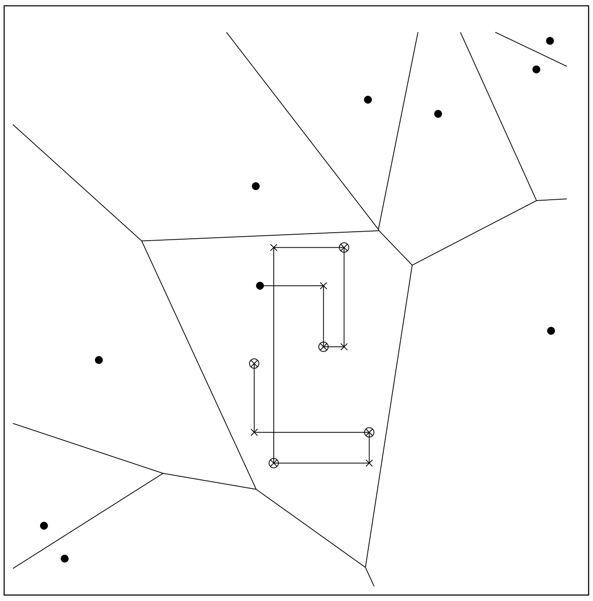


Figure 5.1: A random walk in a Voronoi cell. The models from the previous iterations are marked with a large dot, the intermediate steps with an x and the new models with a circle around the x.

Table 5.1: A summary of the NA algorithm

1. Generate an initial set of n_s models uniformly (or otherwise) in parameter space (the zeroth iteration);
2. Calculate the objective function values for the n_s most recently generated models and determine the n_r models with the lowest objective function value of all models generated so far;
3. Generate n_s new models by performing one uniform random walk in each of the Voronoi cells for the n_r best models (i.e. $\frac{n_s}{n_r}$ models in each cell);
4. Go to 2 until all iterations are done.

Based on the objective function values of the models in the ensemble, the ranking of the models is made. In each of the following iterations, n_s models are uniformly created in the n_r best cells by performing a random walk in each of these cells. The cell decomposition is remade after each iteration, and the sizes and shapes of many cells are thereby changed. One such NA iteration consists of step 2 through step 4 in table 5.1, where a summary of the NA algorithm is found.

When the cell decomposition is remade the chance to find the optimum is increased, since the regions with low objective function values may become parts of Voronoi cells around models with high ranking. The random walks will then have a chance to walk down valleys in function space.

To increase the chances, n_r should be large (of the same size as n_s). With a large n_r, a large amount of cells are re-sized in each iteration and the chance to have included the optimum in one of the new cells is increased. A large n_r results in a more global search. When n_s is large in relation to n_r the search is more local.

Intensification with NA search is shown in figure 5.2. The upper left panel (a) shows the Mexican Hat test problem from above. Here the interval is [-10, 10] for both variables. The global minimum is in the middle (in (0,0)) and the depths of the valleys increase with decreasing radius. Panel (b) shows the Voronoi cells for the randomly chosen models in the initial ensemble. In panel (c), 1200 models have been created by NA search and added to the initial ensemble. Intensifications can be seen close to the global optimum and as vague circles around the optimum. In the last panel (d), the intensification is clear, showing that NA samples the important regions well. In (d) 2600 models in addition to those in (c) have been created with NA search (in total 4300 models).

The Need of Another Control Parameter

The algorithm is designed with only two control parameters, n_s and n_r, but when running the test problems, the size of the initial ensemble, `nsamplei`, showed to be of importance as well. Sambridge does not include this parameter in the papers (see [12] and [14]), but it is included in the NA code.

5.2.1 Control Parameters

The control parameters for the NA algorithm are now specified. For recommended values, see the results.

The parameter `nsamplei` is here defined as a control parameter. The value of the parameter is set to a constant value for all phases (500 for all test problems, except Fallat-Dosso where it is 200). The parameter is evaluated in section 5.3.

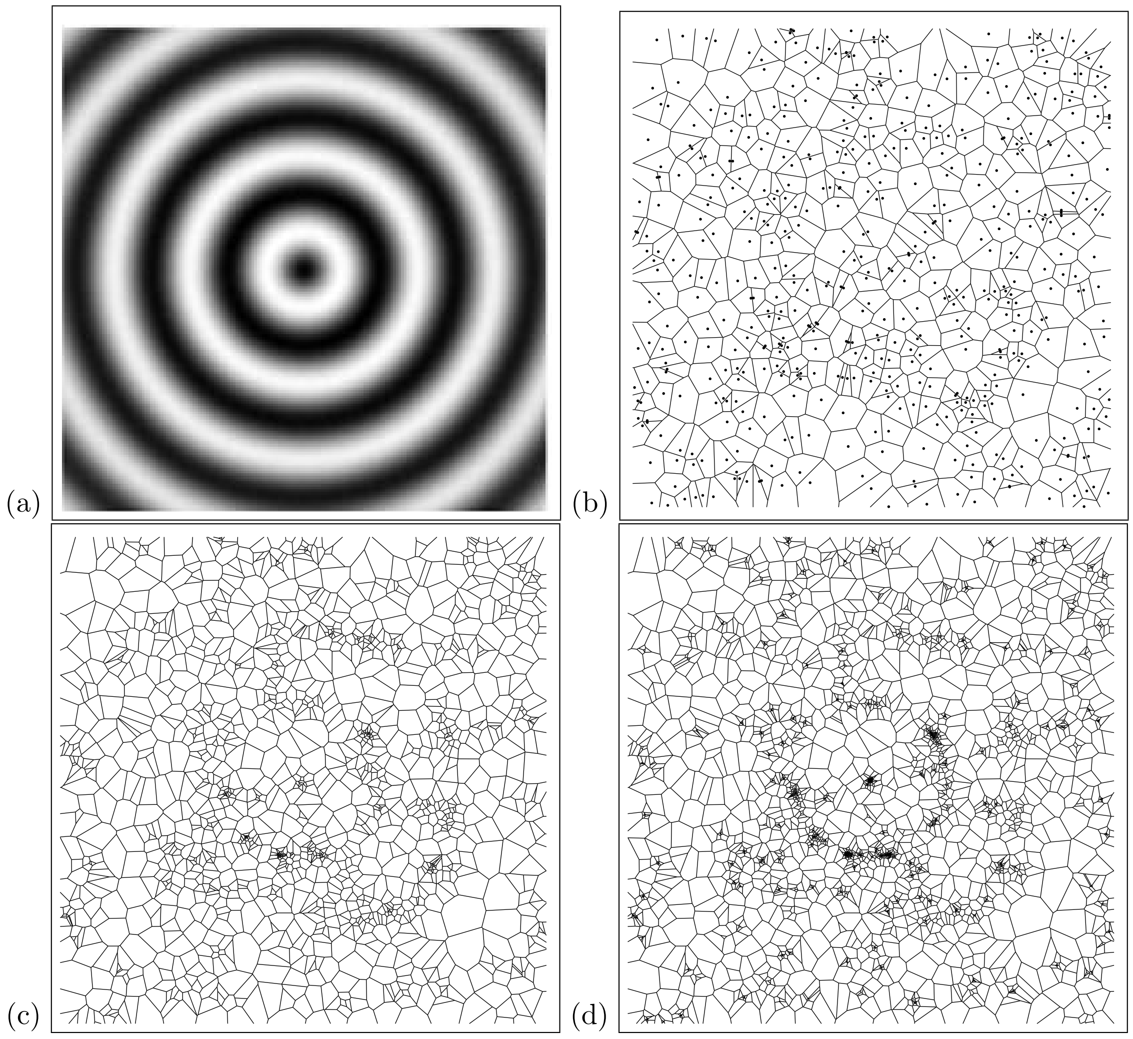


Figure 5.2: Illustration of NA sampling the parameter space. The axes are from -10 to 10 for both variables. (a) The Mexican Hat seen from above (on the interval [-10, 10] for both variables). Black indicates a valley with low objective function values. The global minimum is in the middle. The depths of the circular valleys are increasing with decreasing radius. (b) NA has randomly chosen 500 models, here with their corresponding Voronoi cells. (c) Another 1200 models, in addition to the first 500, have been created by random walks. An intensification in the middle and in a circular fashion around the middle can be seen. (d) Another 2600 models have been created and added to the total ensemble (total 4300 models). Here the intensification is evident in the areas with low objective function values.

- Size of initial ensemble (`nsamplei`)
- Number of models to generate at each subsequent iteration (n_s, positive integer)
- Number of cells to resample at each iteration (n_r, $1 \leq n_r \leq min(n_s, \texttt{nsamplei})$)

5.3 Results

As can be seen in the figures for each of the test problems below, there are curves not reaching as far to the right as the other ones. This depends on the choices of n_s. A larger n_s results in more models in each iteration, and the curves stretch further for each raise in n_s since the number of iterations is the same.

The size of the initial ensemble has been in all cases been chosen to be `nsamplei` = 500 except for Fallat-Dosso where `nsamplei` = 2000.

The values of n_s and n_r have in all cases, except Fallat-Dosso (10^{-5}), been chosen to be 2, 20, 100. The reason for this is to present examples which are easy to compare. This may not include the best choice, but it gives a hint. As can be seen in all figures, different choices of n_s and n_r give different results.

To get statistics, NA has been run 1000 times with different random seeds. The reason why 1000 runs were performed instead of 10000, as for the other three algorithms, is the considerable time consumption of NA.

Mexican Hat

NA does not perform very well in this test problem as can be seen in figure 5.3. The best result for the estimated probability for finding a model below -0.99 is just over 60% with the phase (`nsamplei`, n_s, n_r) = (500, 100, 20), which is a phase with a fast ascent as well. The phase (100, 100) with a comparable result, exhibits a slower ascent. A low value of n_r gives a fast ascent and a high value for n_r a slower ascent.

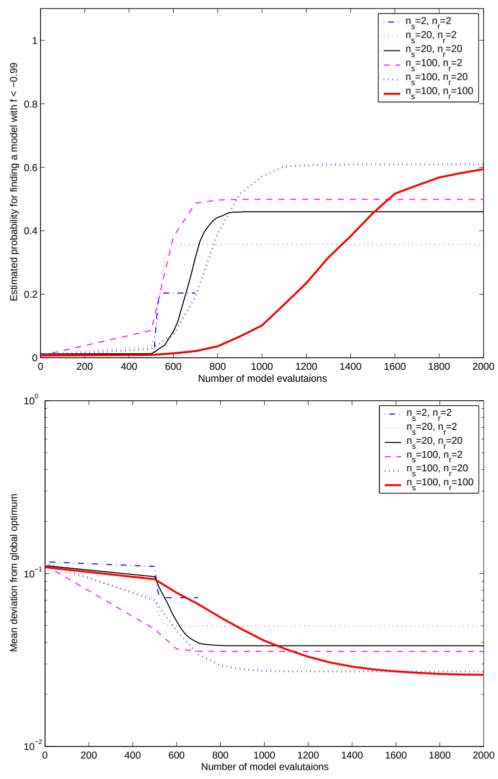


Figure 5.3: Here is the test problem Mexican Hat. As can be seen in the upper panel, the phase (`nsamplei`, n_s, n_r) = (500, 100, 20) gives a good result with a fast ascent.

Fallat-Dosso (success limit = 0.159)

In this case the limit for success is 0.159 and not 10^{-5} as was decided to be the limit for Fallat-Dosso. The results when using the other limit is found in the next section. The limit 0.159 was decided after observing figure 2.2. If the objective function value is below 0.159, the search is likely to have reached the valley of the optimum for all variables.

The best result is just above 70% by the phase (`nsamplei`, n_s, n_r) = (2000, 100, 100) (see figure 5.4). If n_r is lowered (the phases (2000, 100, 2) and (2000, 100, 20)) the greediness of the algorithm is increased. This is seen as the plateaus not reaching as high a value of

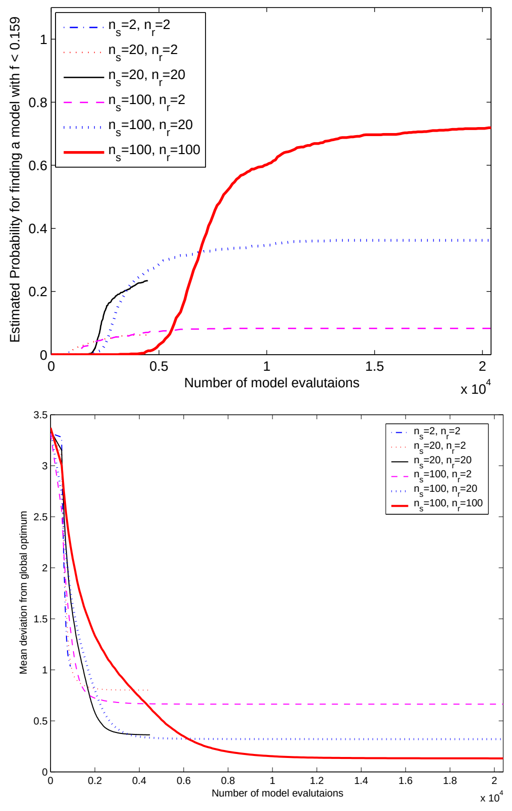


Figure 5.4: This is the test problem Fallat-Dosso. As can be seen in the upper panel, the choice (`nsamplei`, n_s, n_r) = (2000, 100, 100) gives the best result. The limit for success is here 0.159.

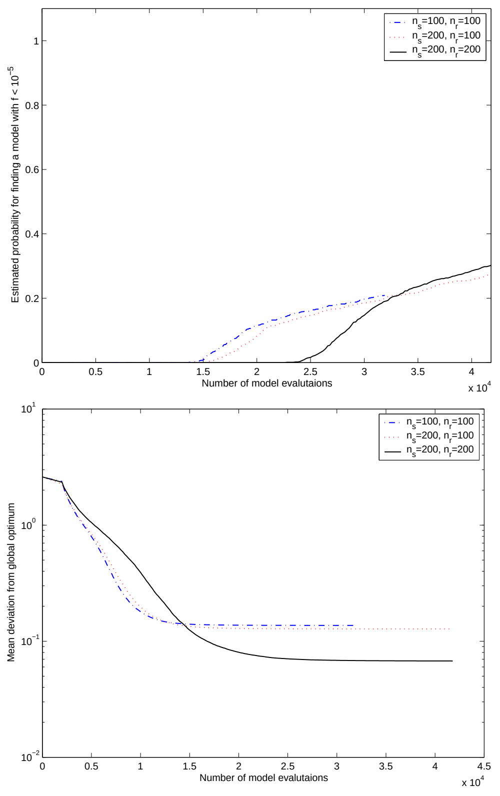


Figure 5.5: This is the test problem Fallat-Dosso with the limit for a successful run equal to 10^{-5}. As can be seen in the upper panel, the choice (`nsamplei`, n_s, n_r) = (2000, 200, 200) gives the best result.

the estimated probability. The phase (2000, 2, 2) does not manage at all in this case.

The test problem has a higher dimension than the other test problems, which makes the size of the search domain a lot bigger. The size of the initial ensemble has therefore been increased and `nsamplei` = 2000.

Fallat-Dosso (success limit = 10^{-5})

The limit for a run to be counted as successful is here 10^{-5}. Observe the different values for n_s and n_r, which are here 100 and 200, since otherwise there would not have been any successful results.

As can be seen in the figure 5.5, increased values for the control parameters give better results. The phase (2000, 200, 200) is best, but the other two phases are not far behind.

The size of the initial ensemble is, as for the other Fallat-Dosso case, `nsamplei` = 2000.

Easom

The phase (`nsamplei`, n_s, n_r) = (500, 100, 2) is the one getting the best results for the Easom test problem with around 90% (see figure 5.6). For each choice of n_r, a low n_s gives a faster ascent than a high value does.

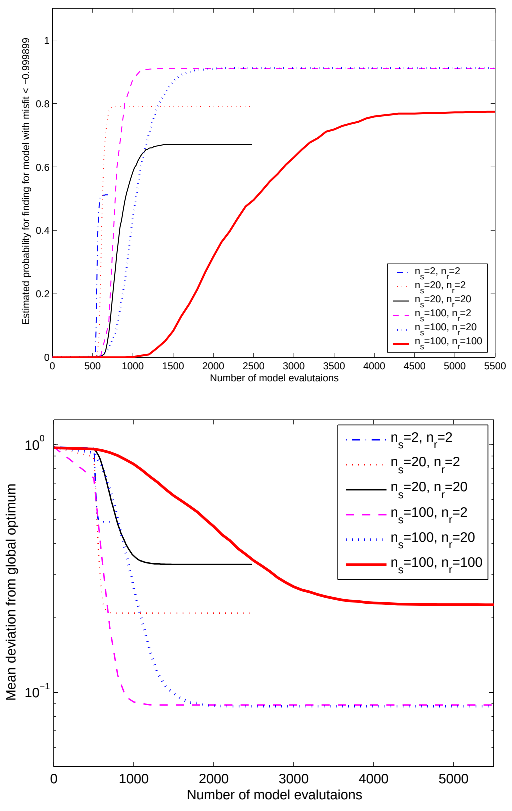


Figure 5.6: NA search for the test problem Easom. As can be seen in the upper panel, the choice (`nsamplei`, n_s, n_r) = (500, 100, 2) gives the best result as well as a fast ascent.

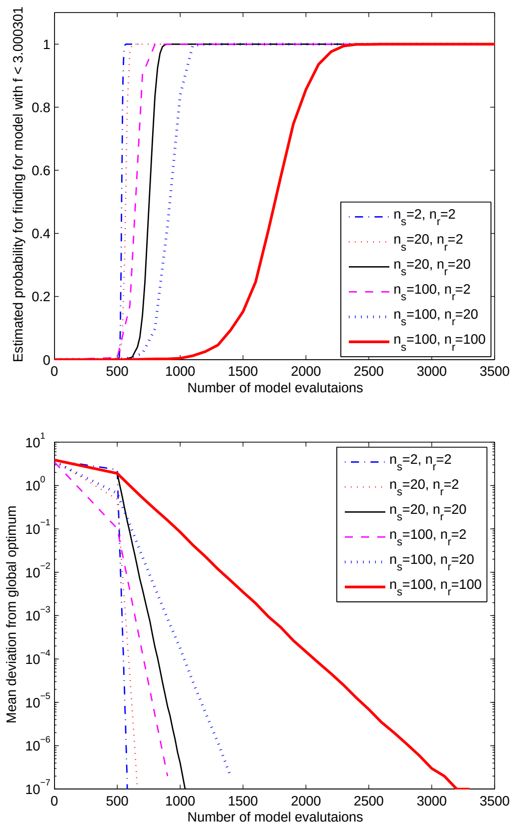


Figure 5.7: As can be seen in the upper panel, the phase (`nsamplei`, n_s, n_r) = (500, 2, 2) reaches 100 % fastest for the test problem Goldstein-Price.

Goldstein-Price

As can be seen in figure 5.7, the phase (`nsamplei`, n_s, n_r) = (500, 2, 2) finds the optimum faster than the other five phases. All phases reach 100% but (2, 2) is ascending fastest. Similar observations can be made as for Easom: a smaller n_r gives a faster ascent and for a given n_r, a smaller n_s gives a faster search.

Shubert

In this test problem, NA succeeds quite well for the phases (`nsamplei`, n_s, n_r) = (500, 2, 2) and (500, 20, 2). They both have fast ascent and reach about 95%. If 100% is required then the phase (500, 20, 20) is the one getting up fastest (see figure 5.8). Also here a similar behaviour as for Easom and Goldstein-Price can be observed.

`nsamplei`

As can be seen for the Mexican Hat problem, there are big differences between the three choices of values of the parameter `nsamplei` (see figure 5.9). The significance of the parameter depends on the problem. For some problems it works all right having the size of

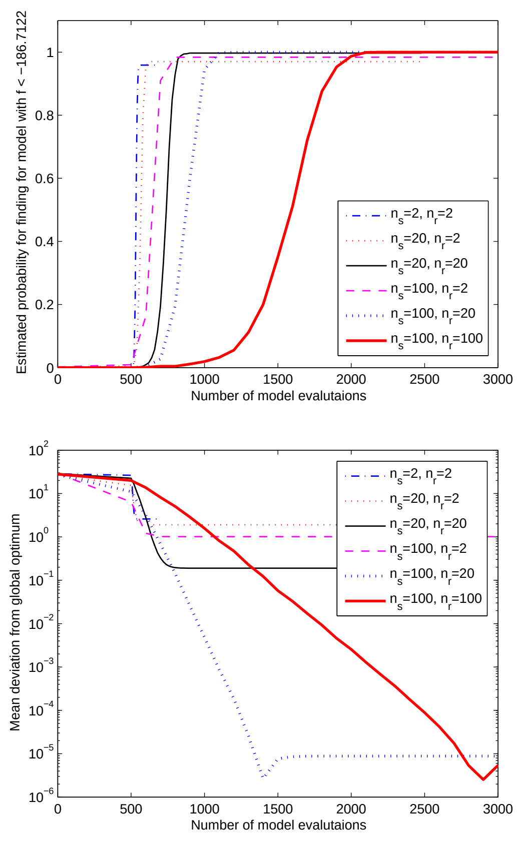


Figure 5.8: In the test problem Shubert (`nsamplei`, n_s, n_r) = (500, 20, 20) is the phase which reach 100 % first, but the phases (500, 2, 2), (500, 20, 2) and (500, 100, 2) are rising faster and are almost as good.

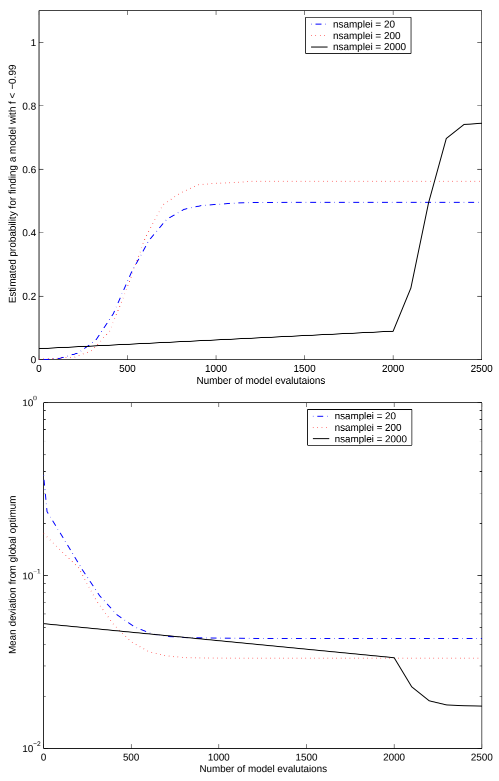


Figure 5.9: Different choices of `nsamplei` give different results in the estimated probability (upper panel) and mean deviation from optimum (lower panel). (`nsamplei`, n_s, n_r) = (500, 100, 20) was the phase with the best result for Mexican Hat (see figure 5.3), and (n_s, n_r) = (100, 20) here too.

the initial ensemble of the same size as n_s, but for more difficult problems as the Mexican Hat, it is of great importance that the initial ensemble has explored the search domain well before beginning the NA search. NA can be a greedy algorithm. If it has not sampled the favourable parts of the search domain from the start, it might choose the wrong cells to search within and never find the global optimum.

Choosing a large `nsamplei` is expensive and therefore it is more to take into account than only good performance.

Some of the results above can probably be improved by increasing `nsamplei`, but there was not enough time to make more tests in this study.

5.4 Discussion

It appears that the value of n_r is very important for how fast the curves for the phases ascend. As can be seen in all test problems, a low n_r gives a fast ascent, but it does also, in most cases, result in a more greedy search. A low n_s also seems to make the search faster and also more greedy, although the behaviour is not as significant and clear as for n_r.

When it comes to the relative importance of n_r and n_s, it seems that n_r is more important for both the ascent and for how greedy the algorithm becomes.

The test problem Mexican Hat is a difficult analytical problem for NA (as can be seen in figure 5.3). There are a lot of valleys in function space to get trapped in, and high hills in between. The algorithm easily gets trapped in a valley, without any ability to check if lower objective function values appear behind the surrounding hills (see figure 2.1).

The fact that NA does not reach 100% in the Fallat-Dosso case can be explained by the number of dimensions. In Fallat-Dosso there are six dimensions instead of two as in the other test problems. Fallat-Dosso is a tricky problem and a lot of exploration is needed. Specifically, `nsamplei` and n_r need to be large, and the algorithm needs to check in a lot of cells for the optimum. This can be illustrated with the phases (2000, 100, 100), (2000, 100, 20) and (2000, 20, 20) in figure 5.4. (2000, 100, 100) reaches the highest value, and the other two phases provide much worse results. The number of models created in each cell, n_s, is not the important parameter. Since (2000, 20, 20) and (2000, 100, 20) are very much alike, the setting of n_r is determining the performance.

Note the difference between the two cases with the two different limits for success for the test problem Fallat-Dosso. When the limit is decreased, the chance for NA to succeed is of course also decreased. When NA has been successful in 70% of the cases after 20000 evaluations for the limit 0.159, the success fraction is less than 20% at 20000 evaluations for the lower limit. For the lower limit, all three phases give results which are quite close to each other.

Sambridge [12] has emphasized the significance of the adaptively changed Voronoi cell structure for finding the optimum, as the iterations proceed. Still, however, the performance is not always good enough according to the results here; the algorithm is too greedy. A way to improve could be to increase `nsamplei` (which is quite expensive) or to develop the algorithm in such a way that it chooses other models than the best ones for the random walks. In the latter case, ideas from Simulated Annealing (see section 3.1.1) could perhaps be used.

Sambridge says that the choice of n_r and n_s is important since the search gets more explorative if they are almost equal and quite large and more local if n_r is small and n_s is large. But Sambridge also says that if you choose n_r = 2 you will get as good results as if you had chosen a larger value, if the algorithm runs long enough [12]. This statement has not been confirmed in the test runs presented in this report.

A larger value of n_s moves the take-off of the success fraction curve to the right, implying a larger number of model evaluations, which makes the run more expensive.

Sambridge says in [12] and [15] that NA was created to find ensembles of models that sample the regions with low objective function values well and not primarily to find a single optimal model. In the articles [12], [15], he concentrated on ensembles of models. But Sambridge also states that although NA is intended for collecting ensembles, it performed just as well as any other approach to the global optimisation in his tests [12]. The results above give a hint of how the control parameters affect the outcome of the runs for optimisation applications.

All four global search techniques are compared and discussed in chapter 7. See also chapter 9, where the algorithms are used on an inverse problem and an ensemble of models is collected.

5.4.1 Tips and Suggestions

The NAB program package is well suited and prepared for parallel computing. Guides to the NA program package can be found in [16] and [14].

Time Consumption

In contrast to the other three optimisation algorithms, the time consumption of NA has a quadratic increase with the number of model evaluations. This can be seen in figure 5.10, which concerns the synthetic test example Mexican Hat for which the CPU time for the actual model evaluations is very small. The quadratic increase arises from the computations of Voronoi cell intersections with axes along which the random walks take place.

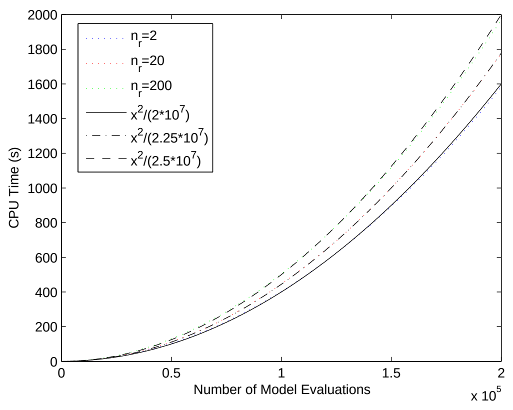


Figure 5.10: The three dotted curves show the CPU time versus number of model evaluations for different values of n_r for the Mexican Hat problem. $n_s = 200$ and `nsamplei` $= 1000$ in each case. A legible quadratic appearance is seen. The remaining three curves show best-fitting quadratic curves.

To see the time consumption of the subroutines in NA, a profiler was run for the Mexican Hat problem. In this way, it was made clear that the three subroutines for the calculations of the Voronoi cell intersections with lines are using almost all time spent (together almost 100%). The time for the actual model evaluations is negligible in this case. When applying NA to a more applied problem, the time consumption for the model evaluations is typically dominant.

Suggestions for Further work

- In NA, only the n_r cells with the lowest objective function values are chosen for the random walks. If there was a probability of choosing other cells with higher objective function values, maybe NA could find the optimum more often. The probability could be decreased with time (cf. Simulated Annealing, section 3.1.1).
- As it is now, the first axis along which a random walk step is supposed to be taken, is randomly chosen. After that, the axes are cycled through until all axes are considered. If the order of the axes was totally random, maybe the algorithm would make an improved search.
- The subroutines concerning the cell intersections are the most time consuming. If most of the points in the search domain were ignored, the calculations in these subroutines would be faster. An idea could be to take into account only the models in a region cut out around the current model, in whose Voronoi cell the algorithm is to walk. The computations time would be reduced, but there is a risk that the resulting Voronoi cell estimate would be larger than the correct Voronoi cell.
- If `nsleep`, the number of direction axes cycles to make before accepting a new model, was a control parameter instead of a fixed number in the code, maybe this could give better results.

Chapter 6

Tabu Search

Rune Westin

6.1 Background

The basics of Tabu Search were developed by Glover in 1977 when he attempted to incorporate artificial intelligence into an optimisation algorithm. The word "tabu" is Tongan - a polynesian language - and indicates something that cannot be touched because it is sacred. In Tabu Search, certain forbidden (or tabu) moves are stored in memory lists to prevent entrapment in cycles. The algorithm is intended for combinatorial problems but attempts have been made to adapt it to continuous problems. I am currently aware of three different variants of Tabu Search for continuous optimisation problems.

Discretised Tabu Search (Directional) is used by Vinther and Mosegaard [17]. Certain move directions are tabu.

Discretised Tabu Search (Model-wise) is used by Michalopoulou and Ghosh-Dastidar [18], [19]. Certain models are tabu. Michalopoulou and Ghosh-Dastidar use the name "Tabu".

Enhanced Continuous Tabu Search is developed by Chelouah and Siarry [1] as an improvement on *Continuous Tabu Search*, developed by Siarry and Berthiau [20]. It is also implemented by Teh and Rangaiah [21].

6.1.1 The Notion of the Current Model

Since Tabu Search searches for optima by moving its attention from model to model according to certain rules, a set of variables keeping track of the current position of the algorithm is needed. The *current model*, in this text denoted $\mathbf{m}^*$, is the point of view for the Tabu Search algorithm at each moment.

6.2 Discretised Tabu Search

As listed above, I have encountered two variants of Tabu Search that discretise the continuous search domain [17], [18], [19]. I call this category of algorithms "Discretised Tabu Search" (DTS).

6.2.1 Discretisation and Movement

Common for both DTS algorithms is the way they discretise the search domain and define the neighbourhoods around each model.

The search domain of N dimensions is discretised with some step δ_n in each dimension $n = 1, \ldots, N$. This means that the search domain is restricted to a *model space* containing

$$\begin{tikzcd}
 & \mathbf{v}'_2 & \arrow[rr, bend left] & & & \mathbf{m}'_2 & \\
\mathbf{v}'_3 & \mathbf{v} \arrow[l, "1"'] \arrow[r, "1"] \arrow[u, "1"'] \arrow[d, "1"] & \mathbf{v}'_1 & & \mathbf{m}'_3 & \mathbf{m} \arrow[l, "\delta_1"'] \arrow[r, "\delta_1"] \arrow[u, "\delta_2"'] \arrow[d, "\delta_2"] & \mathbf{m}'_1 \\
 & \mathbf{v}'_4 & & & & \mathbf{m}'_4 &
\end{tikzcd}$$

Figure 6.1: Definition of the neighbourhood in DTS in a two-dimensional search domain for both the index vector $\mathbf{v}$ and the real vector $\mathbf{m}$. $\mathbf{v}'_i$ denote neighbours of $\mathbf{v}$ and $\mathbf{m}'_i$ denote neighbours of $\mathbf{m}$.

a finite number of models. Every model $\mathbf{m}$ can be represented in an *index space* by a vector $\mathbf{v}$ of N integer indices. A change in v_n by 1 will give a change in the corresponding m_n by δ_n (see figure 6.1).

The *neighbourhood* to $\mathbf{m}$ is defined as all models that can be reached by changing one element of $\mathbf{v}$ by 1 in index space. In model space, this means moving one step δ_n in only one dimension n (see figure 6.1). In other words, diagonal moves are not allowed. Each model has $2N$ neighbours (except for models at the boundaries of the search domain).

At the beginning of the algorithm, an initial model $\mathbf{m}^*$ is either chosen or randomly selected depending on the preference of the user. In each iteration, the neighbours of $\mathbf{m}^*$ are determined. For each neighbour $\mathbf{m}'_n$, the objective function value f'_n is calculated and the neighbour with the lowest objective function value f'_{best} is chosen as the the new current model $\mathbf{m}^*$, even if this leads to a model with a higher objective function value than the previous current model (i.e. $f^*_{\text{new}} > f^*_{\text{old}}$). Thus, DTS can climb from a model with a lower objective function value to a model with a higher objective function value.

6.2.2 Discretised Tabu Search (Directional)

This variant of DTS is used by Vinther and Mosegaard on seismic inversion problems [17]. It distinguishes itself from other variants of Tabu Search by making *directions* for moves tabu. I have therefore used the name "Discretised Tabu Search (Directional)" (or DTS$_\text{D}$ for short) whilst Vinther and Mosegaard use the name "Tabu Search" in their article.

Tabu List

To avoid moving back to previous models and risk getting stuck in cycles, the algorithm keeps track of the latest move directions. The reverse move directions are made tabu (i.e. forbidden) and are stored in a *tabu list* with a length defined by the user.

Whenever a move is to be made, the algorithm checks if the direction of the move giving the lowest objective function value is in the tabu list. If it is, this move is discarded and the move giving the next lowest objective function value is checked, and so on until the best non-tabu move is found. This move is then made, giving a new $\mathbf{m}^*$, and the opposite move direction is recorded in the tabu list.

For example, if a move has been made in the positive n-axis direction, the negative n-axis direction is added to the tabu list. When a new direction is recorded in the tabu list, the oldest direction in the list is removed so that the length of the list is kept constant.

In this way, the tabu list gives the algorithm the ability to climb upwards to higher objective function values and helps it avoid getting trapped in local minima.

Aspiration Criterion

Sometimes, the tabu list prevents the algorithm from moving in a direction that would be desirable to explore. For example, one neighbour of **m*** could have a significantly lower objective function value compared to all previously visited models, while the move to this neighbour from **m*** is forbidden because the direction of the move is in the tabu list.

To avoid missing such interesting models, an exception called the aspiration criterion is introduced. The idea is that the tabu status of a move can be ignored when the move leads to a model with an exceptionally good objective function value.

Termination Criterion

There are several ways to stop the search. One way is to define a small objective function value that must be reached before the search is ended. Another one is to define a maximum number of moves. Often, several different termination criteria are combined.

6.2.3 Discretised Tabu Search (Model-wise)

This algorithm was used and described by Michalopoulou and Ghosh-Dastidar for application to source localisation and geoacoustic inversion [18]. The variant makes certain models forbidden and therefore I call it "Discretised Tabu Search (Model-wise)" (or DTS$_\mathrm{M}$ for short). In their article, Michalopoulou and Ghosh-Dastidar use the name "Tabu".

$$\begin{tikzcd}
0.75 \arrow[r, "1"] & 0.4 \arrow[d, "2"] & 0.55 \arrow[d, shift left, "7"] & 0.6 & 0.7 \\
0.6 & 0.45 \arrow[r, "3"] & 0.3 \arrow[u, shift left, "6"] \arrow[r, shift left, "4"] \arrow[d, "8"] & 0.5 \arrow[l, shift left, "5"] & 0.45 \\
0.7 & 0.55 & 0.6 \arrow[d, "9"] & 0.4 \arrow[d, shift left, "16"] & 0.5 \\
0.7 & 0.4 & 0.2 \arrow[r, "10"] & 0.1 \arrow[u, shift left, "15"] \arrow[r, shift left, "13"] \arrow[d, shift left, "11"] & 0.25 \arrow[l, shift left, "14"] \\
0.65 & 0.45 & 0.15 & 0.2 \arrow[u, shift left, "12"] & 0.35
\end{tikzcd}$$

Figure 6.2: A sample DTS$_\mathrm{M}$ search of 16 moves in a two-dimensional search domain. Notice in moves 2-3 and 8-9 how the algorithm moves over small "ridges". The algorithm can only make *one* move "uphill" in order to find a better model on the other side. If more moves are required it falls back to the previous model.

Tabu Lists

This part of the algorithm differs substantially from the list described in 6.2.2. Here, a move is defined as a direction *and* a model. There are three lists.

Reverse List: This list contains the past K moves that led to an improvement ($K = 2N$ where N is the dimension of the search domain). These moves must not be reversed. In figure 6.2, this is illustrated in for example move 10 which is not reversed.

Forward List: This list contains the past M moves, where M is given by the user. These moves must not be repeated. Move 4 in figure 6.2 is for example not repeated.

Objective function value List: This list contains objective function values for the last evaluated models to avoid unnecessary calculations. The objective function value for the model reached in move 6 in figure 6.2 had already been evaluated when the direction for move 2 was determined.

Expansion

After some time, a model is reached from which all moves are tabu. In this case, a local minimum has been found. This minimum could

$$\begin{tikzcd}
(v_1-k, v_2+k) & (v_1, v_2+k) & (v_1+k, v_2+k) \\
(v_1-k, v_2) & (v_1, v_2) \arrow[l] \arrow[r] \arrow[u] \arrow[d] \arrow[ul] \arrow[ur] \arrow[dl] \arrow[dr] & (v_1+k, v_2) \\
(v_1-k, v_2-k) & (v_1, v_2-k) & (v_1+k, v_2-k)
\end{tikzcd}$$

Figure 6.3: Definition of the $\mathrm{DTS_M}$ expansion neighbourhood in a two dimensional search domain for the index vector $\mathbf{v}$. k is a random integer larger than one.

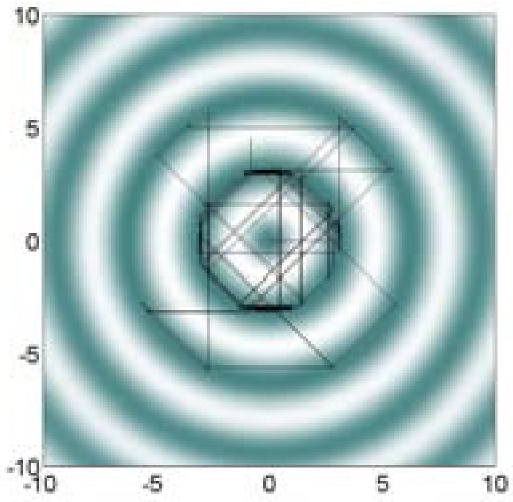


Figure 6.4: A path of the $\mathrm{DTS_M}$ walker minimising Mexican Hat. Darker areas are regions of lower objective function values.

of course be the global minimum but in most cases it is not. In order to escape this minimum and explore other regions, a longer move – called a "jump" – is made.

A new kind of neighbourhood is defined where the neighbours $\mathbf{m}'$ of a model $\mathbf{m}$ are those models that can be reached by changing either one or all elements of the corresponding index vector $\mathbf{v}$ by k where k is a random integer grater than 1 (see figure 6.3). This means that also diagonal moves are allowed. The total number of neighbours in this new neighbourhood will be $2N + 2^N$.

It is sometimes desirable to make a more thorough exploration by defining two more neighbourhoods with increasing size. The second and the third neighbourhood is generated by doubling and tripling k. The neighbour with the best objective function value is chosen as the new current model.

The combined neighbourhood will have a size of $3(2N + 2^N)$ models making this kind of expansion very cumbersome for problems with a large number of dimensions.

6.2.4 Implementation of $\mathrm{DTS_M}$

I decided to test $\mathrm{DTS_M}$ in a Fortran 77 implementation based on the description in [18]. To my knowledge, the only significant deviation from the description that I made was the inclusion of mirroring. This was needed when the algorithm ventured outside the search domain.

I discovered that without expansion, the algorithm was extremely greedy and would not climb more than one step upwards, which would seldom allow it to escape local minima. The consequence was that an expansion was needed every time the algorithm was trapped. Figure 6.4 shows one of the few successful minimisations of Mexican Hat. Many expansions were performed before a sufficiently good starting point was found for the local search. It should be pointed out that the distances covered by the local searches are only a fraction of the distances covered in the expansions.

The conclusions from these premises were:

- $\mathrm{DTS_M}$ is a fairly fast local optimiser.
- Global optimisation requires many expansions.
- One expansion requires $3(2N+2^N)$ objec-

tive function evaluations, making minimisation of functions of many variables very costly.

Considering that the applications foreseen for our optimisation algorithms (see chapter 8) have 10 to 24 dimensions, I decided to search for other variants of Tabu Search for continuous optimisation problems.

6.3 Enhanced Continuous Tabu Search

The previous variants of Tabu search discretise the search domain for a convenient definition of neighbourhoods. This also simplifies the management of tabu lists that contain previously visited models. It is, however, sometimes useful to have a higher resolution and flexibility than that which can be obtained with a reasonable discretisation.

Siarry and Berthiau proposed an adaptation where models could be chosen from anywhere in the continuous search domain in a procedure called "Continuous Tabu Search" (CTS) [20]. Later, Chelouah and Siarry improved CTS by redefining the neighbourhood and adding an exploration stage. The resulting algorithm is called "Enhanced Continuous Tabu Search" (ECTS) [1].

6.3.1 Neighbourhood

The neighbourhood of a model $\mathbf{m}^*$ in CTS consists of η neighbours $\mathbf{m}'_i$ where η is twice the number of dimensions up to a maximum of 10 ($\eta = \min[2N, 10]$). To determine the neighbours, we consider a set of $\eta + 1$ concentric spheres centred on $\mathbf{m}^*$. The spheres have radii $h_0, h_1, \ldots, h_\eta$ where $h_0 < h_1 < \ldots < h_\eta$. A neighbour $\mathbf{m}'_i$ is placed randomly between the shells of two spheres so that $h_{i-1} \leq ||\mathbf{m}^* - \mathbf{m}'|| \leq h_i$ where $||\ldots||$ denotes the Euclidian norm.

The partitioning of the neighbourhood depends on three parameters: the radius of the outer sphere h_η, the radius of the inner sphere h_0 and the number of neighbours η. Three partitioning methods for h_i are proposed:

1. *Geometrical partitioning.* Each radius is half of the next larger radius (see figure 6.5). This promotes placing neighbours close to $\mathbf{m}^*$.

$$h_{\eta-i+1} = \frac{h_\eta}{2^{i-1}}, \quad i = 1, \ldots, \eta \tag{6.1}$$

2. *Linear partitioning.* The distances between shells are equal (see figure 6.5).

$$h_{\eta-i+1} = \frac{h_\eta(\eta - i + 1)}{\eta}, \quad i = 1, \ldots, \eta \tag{6.2}$$

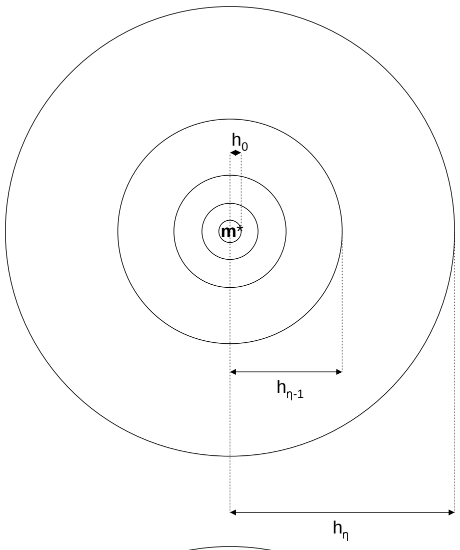
3. *Isovolume partitioning.* The volumes between the shells are equal. Here, the preset h_η has no consequences for the partitioning. The outermost shell radius will be calculated from h_0 and η.

$$h_{i+1} = \left(h_i^N - \frac{h_i^N}{\eta}\right)^{\frac{1}{N}}, \quad i = 0, \ldots, \eta - 1 \tag{6.3}$$

When developing ECTS, Chelouah and Siarry decided to replace the hyperspheres with hypercubes[1] (see figure 6.6) because it is practically easier to select a uniformly distributed random model in a cubical than in a spherical shell. h_i represents the half-length of the sides of hypercube i.

Teh and Rangaiah have proposed a convenient method for selecting the neighbours [21]. I have reformulated the description in their article slightly:

[1]In [1], Chelouah and Siarry refer to the neighbourhood as hyperrectangular but since all sides of the hyperrectangles are equal, I find it more suitable to use the term hypercube.



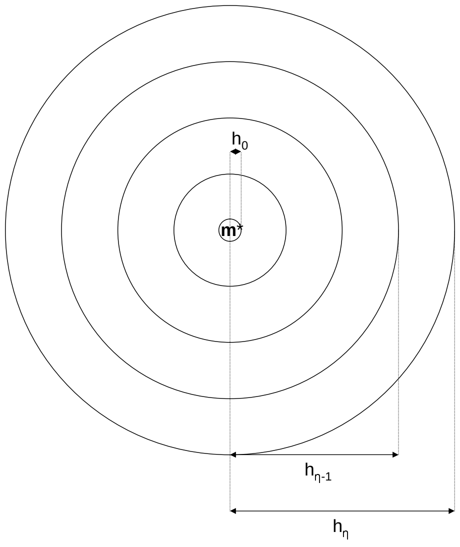


Figure 6.5: Different partitioning methods in CTS. Geometrical partitioning (top) and linear partitioning (bottom). Geometrical partitioning promotes neighbours close to $\mathbf{m}^*$.

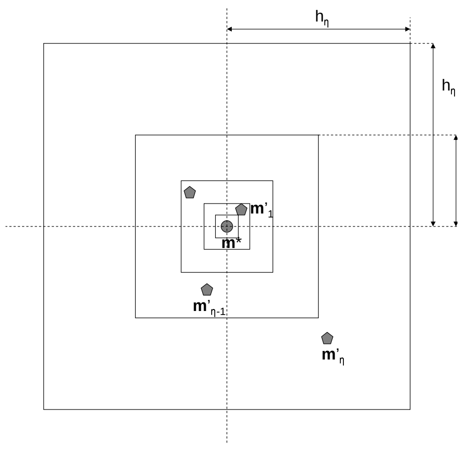


Figure 6.6: Geometrical partitioning in ECTS. The hyperspheres of CTS have been replaced by hypercubes.

1. Calculate h_i with the selected partitioning so that $h_0, h_1, \ldots, h_\eta$ are known. Set $i = 0$.

2. Update $i = i+1$ with the intention of generating a neighbour $\mathbf{m}'_i$ between the shells of hypercube i and hypercube $i-1$.

3. Evaluate the upper bounds $U_i^n = m_n^* + h_i$ and lower bounds $L_i^n = m_n^* - h_i$ for hypercube i, where $n = 1 \ldots N$ is the dimension and m_n^* is the n-component of $\mathbf{m}^*$.

4. Generate a uniformly random model $\mathbf{x}_i = (x_i^1, x_i^2, \ldots, x_i^N)$ within the bounds, using N rectangularly distributed random numbers r_n in the interval [0,1] and

$$x_i^n = L_i^n + r_n(U_i^n - L_i^n), \quad n = 1, 2, \ldots, N \tag{6.4}$$

5. Calculate the shortest distances α_i^n between the components of the generated model $\mathbf{x}_i$ and the lower bounds L_i^n, given by $\alpha_i^n = x_i^n - L_i^n$.

6. If $(h_i - h_{i-1} < \alpha_i^n < h_i + h_{i-1})$ is fulfilled for all $\alpha_i^n, n = 1, 2, \ldots, N$, then $\mathbf{x}_i$ lies

within hypercube $i-1$ and must be regenerated from step 4. Otherwise, $\mathbf{m}'_i = \mathbf{x}_i$.

7. Repeat steps 2 to 6 until all neighbours have been generated.

6.3.2 Tabu and Promising Lists

ECTS has two lists keeping track of specific locations in search space, the *promising list* for especially interesting regions and the *tabu list* for forbidden regions. These regions are defined as balls centred on points (models) recorded in the list. The balls have radii given by the algorithm, one radius for the promising balls and one for the tabu balls. To find out if a model $\mathbf{m}$ lies within a ball, the Euclidian norm $||\mathbf{m} - \mathbf{x}||$, where $\mathbf{x}$ is the centre of the ball, is compared to the radius of the ball.

6.3.3 Outline of ECTS

ECTS consists of five stages: *Parameter setting*, *diversification*, *search for the most promising region*, *intensification* and *output*. The algorithm is outlined in tables 6.1 and 6.2, and its stages are described more thoroughly in what follows.

Parameter Setting

Some parameters are given by the user. These include the sizes of the tabu and promising lists (N_t and N_p), the search domain and possibly the starting point. Others are calculated from the given parameters, for example the initial radii of the tabu and promising balls (h_t and h_p) and the size of the hypercubical neighbourhood (h_η and h_0). A complete guideline for the parameters are given by Chelouah and Siarry [1].

The user can also select the partitioning methods to be used at the different stages.

Table 6.1: First part of a summary of the ECTS algorithm. Continued in table 6.2.

1. **Parameter setting:** Some parameters are given by the user and some are calculated. A starting model is defined.
2. **Diversification:** A number of promising regions are detected.
   (a) η neighbours to the current model are generated.
   (b) If a promising model is detected, it is recorded in the promising list.
   (c) The best neighbour is taken as the current model and the previous model is recorded in the tabu list.
   (d) Steps 2a-2c are repeated until the stopping criteria are met.
3. **Search for the most promising region:** The most promising region is detected.
   (a) All promising models worse than the average objective function value in the promising list are removed.
   (b) The tabu ball radius and the neighbourhood size are halved.
   (c) η neighbours are generated for each promising model. A promising model is replaced by its best neighbour if the neighbour is better.
   (d) The worst promising model is removed.
   (e) Step 3b-3d are repeated until only one promising model remains.

Table 6.2: Second part of a summary of the ECTS algorithm. Continued from table 6.1.

4. **Intensification:** The most promising region is more thoroughly searched.
    (a) The most promising model is taken as the current model.
    (b) η neighbours to the current model are generated.
    (c) The best neighbour is taken as the new current model and the previous model is recorded in the tabu list.
    (d) Steps 4b-4c are repeated until a specific number of iterations without an improvement have passed.
    (e) The tabu ball radius and the neighbourhood size are halved. The best model yet is taken as the current model.
    (f) Steps 4b-4e are repeated until the stopping criteria are met.
5. **Output:** The best model found is presented.

Diversification

The aim of this stage is to make a wide search in order to find the most promising regions of the search domain. At the beginning of the stage, the tabu list is emptied and the promising list is filled with random models. A start model $\mathbf{m}^*$ is either randomly selected or given by the user.

A neighbourhood to $\mathbf{m}^*$ is defined according to section 6.3.1 and η neighbours are generated. Neighbours within tabu balls or outside the search domain are regenerated until all neighbours are non-tabu and within the search domain. The objective function values for all neighbours are evaluated and the neighbour with the lowest objective function value is chosen as the new current model $\mathbf{m}^*$, even if it is worse than the previous one. New neighbours for the new $\mathbf{m}^*$ are generated and so on. Whenever a new model is chosen, another *iteration* has passed.

When a new current model $\mathbf{m}^*$ has been chosen, the previous one is added to the tabu list. If the tabu list is full, the oldest model in the list is removed to make place for the new model.

The objective of the diversification stage is to fill the promising list with interesting models, preferably separated from each other to avoid redundancy. A value called the "threshold" is equal to the mean value of the objective function values for all models in the promising list. A visited model $\mathbf{m}^*$ is inserted in the promising list if it fulfills three criteria:

- The objective function value of $\mathbf{m}^*$ must be better than the threshold.
- All neighbours of $\mathbf{m}^*$ must have higher objective function values than $\mathbf{m}^*$.
- $\mathbf{m}^*$ must not lie close to an already existing promising model (i.e. within a promising ball).

If a model $\mathbf{m}^*$ fulfills these criteria, the worst model in the promising list is removed and $\mathbf{m}^*$ is inserted in its place. The new threshold is calculated.

The diversification stage continues until a given number of successive iterations have passed without detecting a new promising region and also without finding a new best model $\mathbf{m}_{best}$[2].

Search for the Most Promising Region

This stage is performed to narrow down the list of promising regions to one single promising region. First, all models with an objective function value higher than the threshold are removed from the list. The remaining models in the list are dealt with in an iterative manner:

1. The radii of the tabu balls (h_t) and the size of the hypercubic neighbourhoods (h_η and h_0) are halved.
2. Neighbourhoods are defined and neighbours are generated for all promising models in the list.
3. Each promising model is replaced by its best neighbour, but only if the objective function value is lowered.
4. The worst model in the promising list is removed.
5. Steps 1-4 are repeated until only one promising model remains.

The remaining model in the promising list is taken as the new current model $\mathbf{m}^*$.

Intensification

This stage intensifies the search for the most promising region. The ball with radius h_p around the remaining promising model is taken as the new search domain[3,4].

At the beginning of the stage, the tabu list is emptied.

A new search is initiated as in the diversification stage with the exception that the promising list is ignored; a neighbourhood is defined and η neighbours are generated and checked against the tabu list. The best neighbour is taken as the new $\mathbf{m}^*$, even if it is worse than the previous current model. In each iteration, the algorithm keeps track of the best model found so far $\mathbf{m}_{best}$.

When a predetermined number of successive iterations have passed without an improvement of $\mathbf{m}_{best}$, the radius of the tabu balls (h_t) and the size of the hypercubic neighbourhoods (h_η and h_0) are halved and the tabu list is emptied. The best model found so far is used as the new current model ($\mathbf{m}^* = \mathbf{m}_{best}$) and the search continues from there.

There are two stopping criteria for the intensification stage.

- A predetermined number of successive reductions of the tabu balls and the hypercubic neighbourhoods have passed without detection of a new $\mathbf{m}_{best}$.

[2]The second test was not included in the original ECTS as described in [1]. I added it after experiencing that the algorithm sometimes would reach a really good model at the end of the diversification stage without recording it in the promising list because it did not fulfill the criteria.

In my implementation, I also found it useful to finish the stage only if the last iteration was from a model with a lower objective function value to a model with a higher objective function value.

[3]It is possible that this is an unnecessary restriction of the search domain since the neighbourhoods have become very small and the algorithm is unlikely to move large distances. However, if it does move large distances, this should not have a negative impact on the search since it is always the best model that counts. In my implementation, I kept the original search domain.

[4]It is, as in the case of the neighbourhood in section 6.3.1, convenient to use a hypercube instead of a hypersphere as the new search domain.

- A predetermined number of iterations have passed since the beginning of the intensification stage[5].

Output

At this stage, $\mathbf{m}^*$ is presented to the user along with its objective function value and relevant statistics.

6.3.4 Control Parameters

There are a lot of parameters that can be tuned. Chelouah and Siarry have pointed out some guidelines [1] that should work for problems with several different numbers of dimensions. The control parameters are presented in the following list with Cheloah and Siarry's proposals for parameter values (N is the number of dimensions and δ is the shortest side of the hyperrectangular search domain).

- Tabu list size ($N_t = 7$).
- Promising list size ($N_p = 10$).
- Maximum number of successive iterations without detection of a new promising region in the diversification stage ($\text{Max}_p = 2N$).
- Maximum number of successive iterations without detection of a new best model in the intensification stage ($\text{Max}_b = 5N$).
- Maximum number of successive reductions of the hypercubic neighbourhood in the intensification stage ($\text{Max}_r = 2N$).
- Maximum number of iterations in the intensification stage ($\text{Max}_i = 50N$).
- The initial radius of the tabu balls ($h_t = \frac{\delta}{100}$).
- The initial radius of the promising balls ($h_p = \frac{\delta}{50}$).
- The initial size of the hypercubic neighbourhoods ($h_\eta = \frac{\delta}{5}$).
- The number of neighbours in a neighbourhood ($\eta = \min[2N, 10]$).

All these parameters can of course be varied. In my study, I limited myself to varying N_t, N_p, Max_p, Max_b, Max_r and Max_i. I later discovered that increasing N_p or Max_r did not improve my results significantly and that Max_i only needed to be increased if the algorithm was stopped while the results were still improving. This means that I was down to three parameters for variation: N_t, Max_p and Max_b.

6.4 Results

I tested ECTS on the five synthetic test problems: Mexican hat, Fallat-Dosso, Easom, Goldstein-Price and Shubert. Each problem was optimized with five different parameter settings (i.e. phases) and the number of runs (n_{run}) in each phase was 10000. One of the parameter settings tested is the recommended one from [1] (phase one in each graph), three are deviations from this setup by one parameter (phases two through four) and one is a combination of deviations in all parameters (phase five). The estimated probability for success for the different phases was observed according to equation (2.3), and the mean deviation from the global optimum was calculated according to equation (2.4).

Mexican Hat

The results are displayed in figure 6.7. In the best parameter setting, ECTS succeeded

[5]Again, I found it useful to let the algorithm continue until all neighbours of the current $\mathbf{m}^*$ had worse objective function values than $\mathbf{m}^*$, even if this criterion was fulfilled.

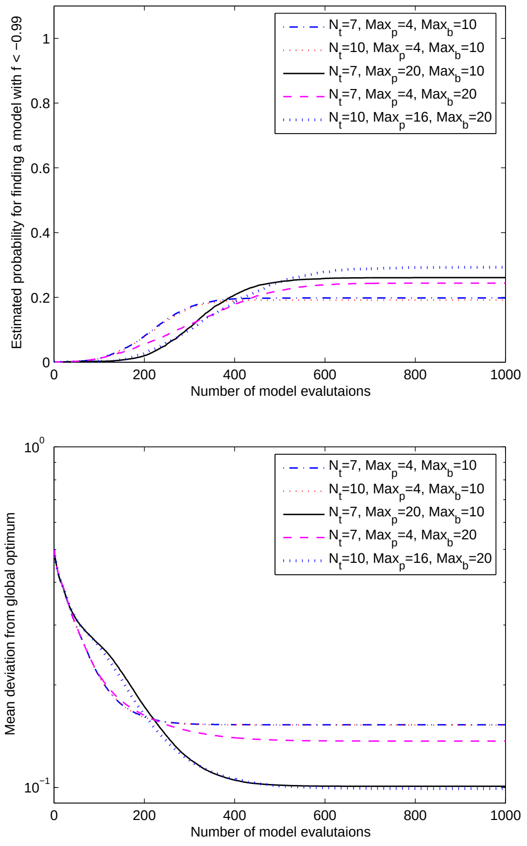


Figure 6.7: ECTS optimisation of Mexican hat. *Top*: The estimated success probabilities. *Bottom*: The mean deviations from the global optimum.

in finding an objective function value less than -0.99 in approximately 30 % of the runs.

Three of the phases are slightly more reliable than the one with the recommended parameter values. One is the phase where the maximum number of successive iterations without detecting a new promising region (Max_p) is increased. This prolongs the diversification stage, giving ECTS a better chance of finding a good promising region to search more thoroughly, at the expense of making the algorithm slower.

The second more reliable phase is the one

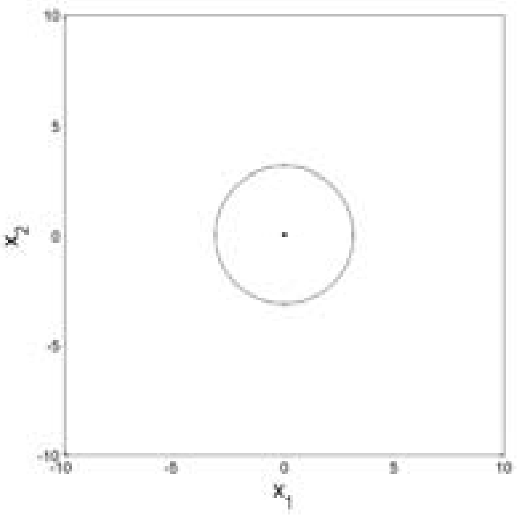


Figure 6.8: The regions where Mexican hat has an objective function value less than -0.99 are marked in black.

where the maximum number of successive iterations without detecting a new best objective function value (Max_b) is increased. I believe that this improvement corresponds to the choice of limit for success in Mexican hat. There are two regions for Mexican hat with an objective function value less than -0.99, one immediately around the origin and one in the circular valley closest to the origin (see figure 6.8). Both of these regions require good local optimisation capability to be found, and increasing Max_b increases this capability in the intensification phase.

The last phase causing improvement involves a combination of prolonged diversification and intensification. This means that the optimisation in this phase is generally more reliable but also more slow than in the other phases in this test.

Fallat-Dosso

Figure 6.9 illustrates that ECTS did not perform well on Fallat-Dosso. The plot of mean

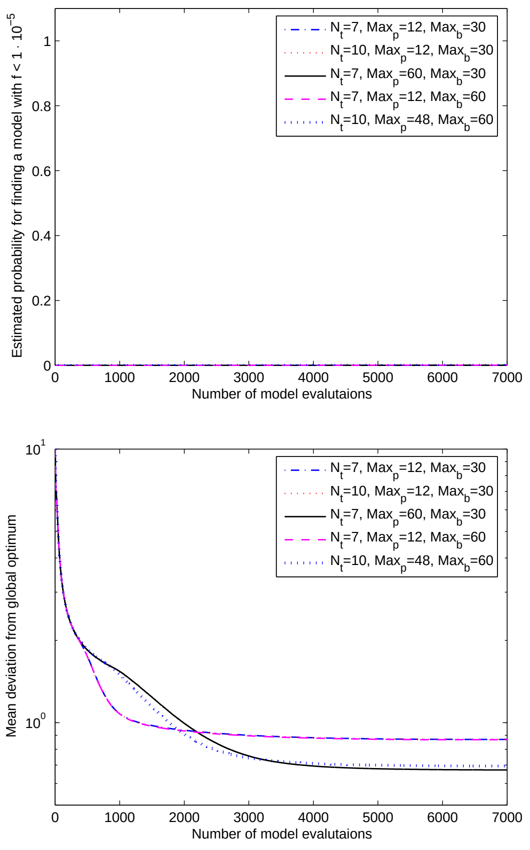


Figure 6.9: ECTS optimisation of Fallat-Dosso. *Top*: The estimated success probabilities. *Bottom*: The mean deviations from the global optimum.

values shows that phases with prolonged diversification stages generally find models with smaller objective function values.

Easom

The results are shown in figure 6.10. Here, phases with a prolonged intensification stage are much less successful in finding the global optimum than the other phases.

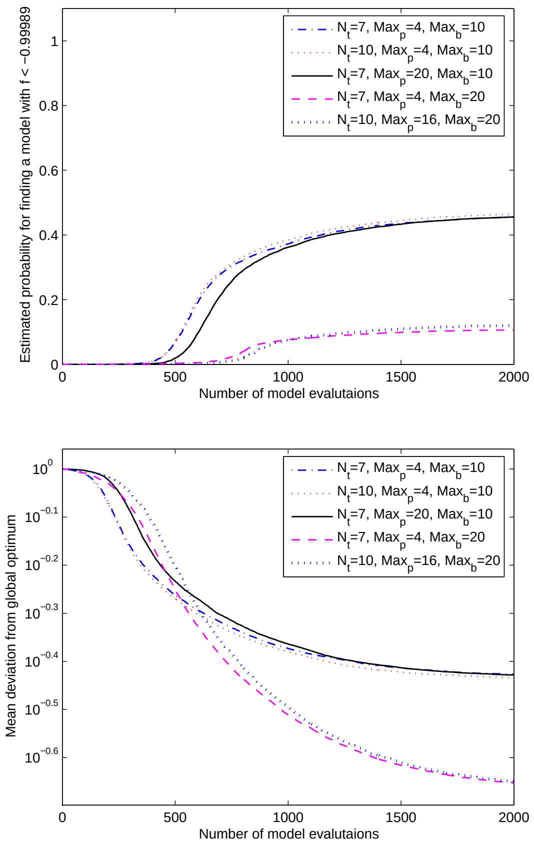


Figure 6.10: ECTS optimisation of Easom. *Top*: The estimated success probabilities. *Bottom*: The mean deviations from the global optimum.

Goldstein-Price

ECTS works fairly well on Goldstein-Price as seen in figure 6.11. There is a small trade-off between speed and reliability, since a prolonged diversification stage increases the probability of success but slows the process down.

Shubert

It should be noted that there are many global optima in the search domain of Shubert. The estimated probability of finding one is illustrated in figure 6.12. Here, it is clear that an increase in Max_p as well as in Max_b improves

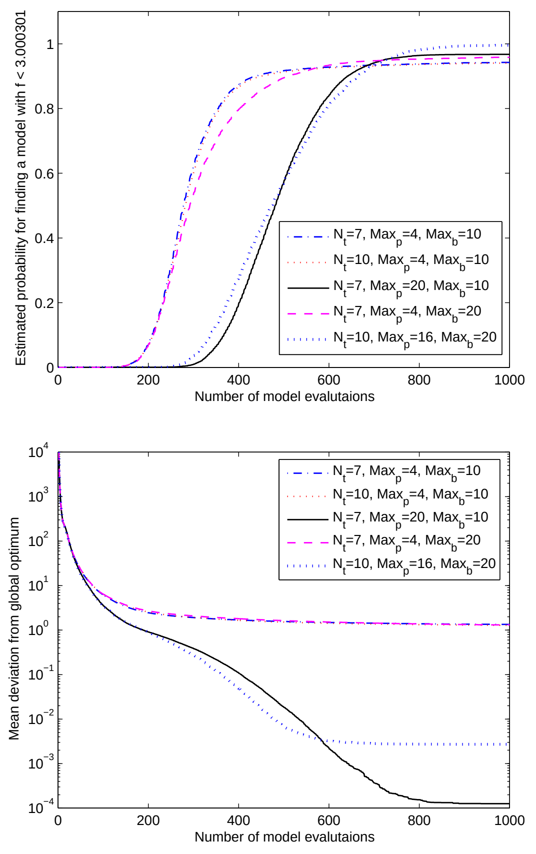


Figure 6.11: ECTS optimisation of Goldstein-Price. *Top*: The estimated success probabilities. *Bottom*: The mean deviations from the global optimum.

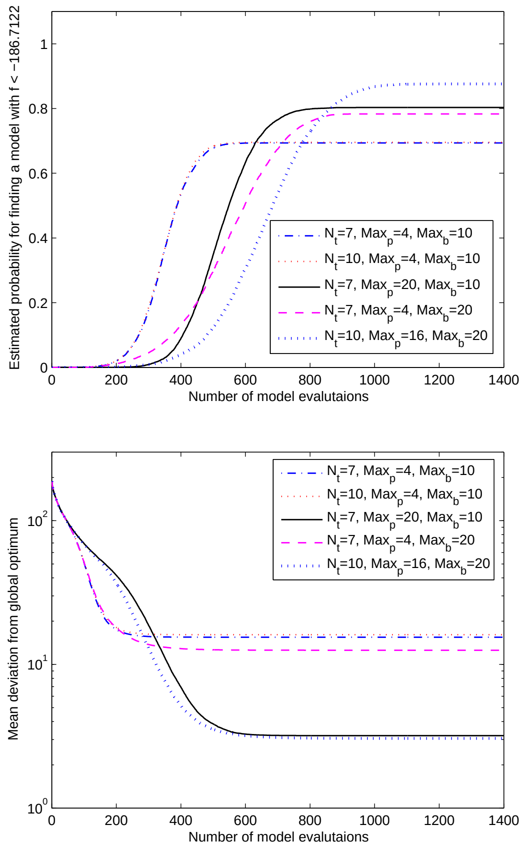


Figure 6.12: ECTS optimisation of Shubert. *Top*: The estimated success probabilities. *Bottom*: The mean deviations from the global optimum.

the chances of success while slowing the process down.

6.5 Discussion

There are both advantages and disadvantages of ECTS. The main advantage is speed. ECTS uses a comparably small amount of objective function evaluations. Another advantage is that ECTS is a good local optimiser, although it is perhaps not as good as designated local optimisers.

The main disadvantage is that ECTS is not particularly reliable. The optima that ECTS finds are sometimes global but in most cases they are not. In Goldstein-Price, the objective function has only three local minima, with an easy, mostly downhill, path to the global minimum. In Shubert, there are many global minima to choose from, improving ECTS's chances of finding a good starting point for its intensification.

An observation is that ECTS finds the global minimum fast if it finds a sufficiently good promising region in the diversification phase. Therefore, attempts to improve the objective function value should be focused on setting the control parameters to prolong the diversification stage.

One of the most puzzling results is ECTS' low probability in finding the global optimum of Easom when using a prolonged intensification stage (see figure 6.10). I have not been able to understand why this behaviour occurs.

Throughout my tests, I could not reproduce the outstanding results presented in the original ECTS article [1]. One observation made by Headar and Fukushima [8] is that there might be inconsistencies in some of the results of Chelouah and Siarry [1]. The success fractions for some of the test functions reported were 100 % even though the average error – given as the difference between the found optimum and the analytical optimum – was greater than the criterion for success.

My conclusion is that although ECTS may not be as reliable as the other optimisation techniques in the tests, it should not be overlooked. If sufficiently low minima are enough and if speed is a major issue, ECTS might be a good choice. There is potential in the algorithm and if improvements were to be done on the diversification stage, increasing its reliability, the speed of ECTS would be a competitive factor. However, at present, the performance of ECTS is not good enough compared to algorithms such as Adaptive Simplex Simulated Annealing and Differential Evolution.

6.5.1 Other Implementations

During my tests, I came upon one article describing an implementation of ECTS for application to phase equilibrium calculations [21]. The authors, Teh and Rangaiah, had used ECTS for diversification and the search for the most promising region. The intensification stage was performed by local optimisers (a modified Simplex method and a quasi-Newton method).

6.5.2 Tips and Tricks

ECTS cannot normally handle problems where the search domain in one or more dimensions has the same upper bound as lower bound. This might be a problem in cases where one would like to keep one variable in the search constant by limiting its domain to just one value. The reason for this problem is that ECTS bases the size of the neighbourhood on the extent of the least extensive search domain dimension δ. If the extent is zero, the size of the hypercubic neighbourhood will be zero and the algorithm gets stuck in the initial point.

In order to circumvent this problem, I defined a search space for the algorithm where the extent in all dimensions is $[0, 1]$. I then created a shell around the objective function routine which transformed the search space of the algorithm to the search space of the objective function. In this way, I could also handle problems where different dimensions have different scales.

Chapter 7

Algorithm comparison

Hanna Gothäll and Rune Westin

In this chapter, the performances of the four optimisation techniques are compared.

7.1 Comparison of Results

For each of the different synthetic test problems, the best phase of each technique has been chosen. The corresponding results for the different techniques are plotted together. The control parameters of the displayed phases are shown in the following manner (the pages where the control parameters are described are shown within parentheses):

Algorithm	Control parameters
ASSA (p. 20)	(`t0fakt` / T_0, β, S, `nPert`, s, `nPertfactor`)
DE (p. 34)	(NP, F, CR)
ECTS (p. 57)	(N_t, Max_p, Max_b)
NA (p. 40)	(`nsamplei`, n_s, n_r)

where `t0fakt` is used in all cases in ASSA, except in the test problem Easom where T_0 is used instead.

Mexican Hat

The best results for each optimisation technique on Mexican hat are shown in figure 7.1. It can be seen that DE and ASSA reach success fractions of 100% with DE being slightly faster

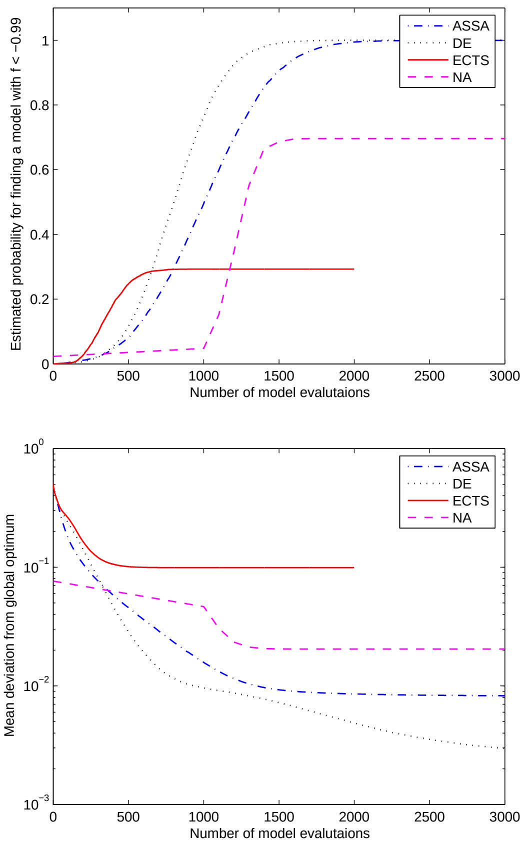


Figure 7.1: A comparison of optimisations of Mexican hat. Phases chosen are: ASSA: (20.0, 0.95, 10, 5, 1.0, 5), DE: (20, 0.5, 0.9), ECTS: (10, 16, 20) and NA: (1000, 100, 20).

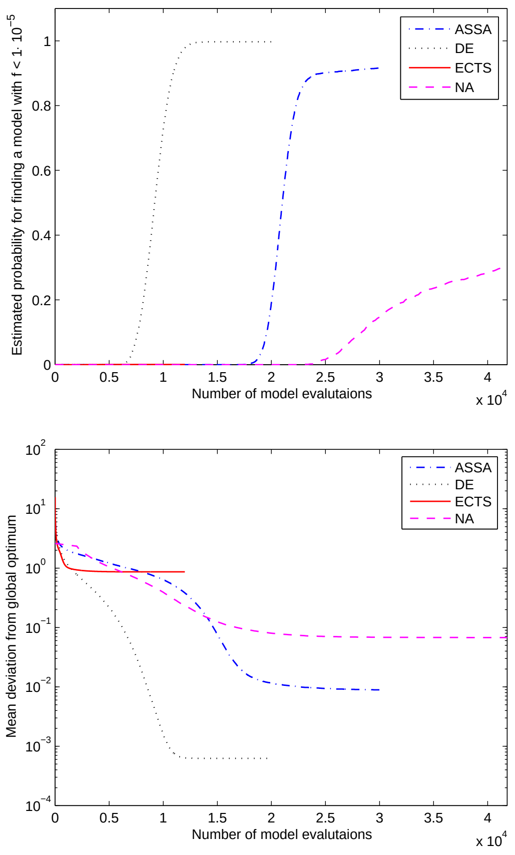


Figure 7.2: A comparison of optimisations of Fallat-Dosso. Phases chosen are: ASSA: (10.0, 0.975, 30, 10, 1.0, 10), DE: (30, 0.5, 0.5), ECTS: (7, 12, 60) and NA: (2000, 200, 200)

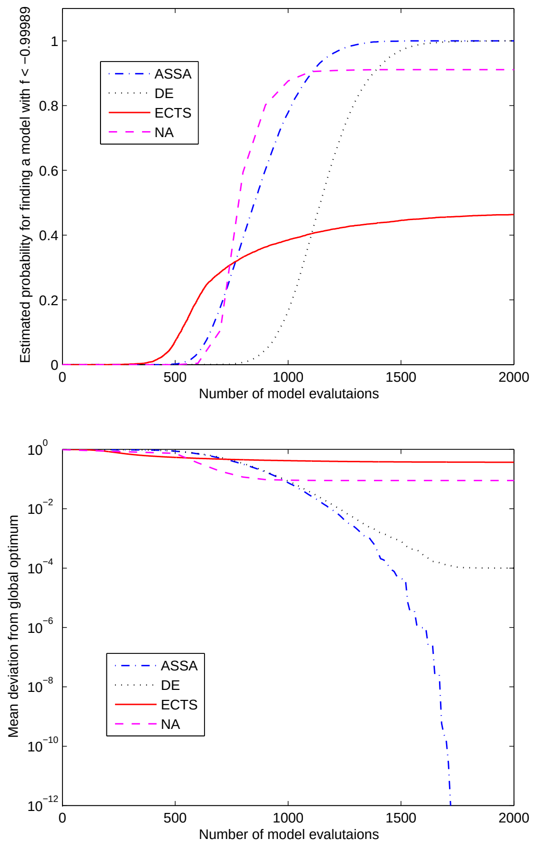


Figure 7.3: A comparison of optimisations of Easom. Phases chosen are: ASSA: (10.0, 0.8, 20, 1, 1.0, 5), DE: (20, 0.5, 0.9), ECTS: (10, 4, 10) and NA: (500, 100, 2)

than ASSA. The fastest ascent is exhibited by ECTS, but it only reaches a success fraction of approximately 30 %. NA finds the required optimum in 60 % of the runs.

Algorithm	Control parameters
ASSA	(20.0, 0.95, 10, 5, 1.0, 5)
DE	(20, 0.5, 0.9)
ECTS	(10, 16, 20)
NA	(1000, 100, 20)

Fallat-Dosso

The Fallat-Dosso results are shown in figure 7.2. DE is by far the most successful algorithm in optimising Fallat-Dosso, regarding speed as well as attained success fraction. ASSA reaches 90 % after approximately 25000 evaluations and NA reaches 30 % after 41000 evaluations. ECTS gets trapped in local minima in almost all runs.

Algorithm	Control parameters
ASSA	(10.0, 0.975, 30, 10, 1.0, 10)
DE	(30, 0.5, 0.5)
ECTS	(7, 12, 60)
NA	(2000, 200, 200)

Easom

For the optimisation of Easom, ASSA and DE perform best of the four algorithms (see figure 7.3), with ASSA somewhat faster than DE. NA, with a success rate of 90 %, is slightly faster than ASSA. ECTS is fastest at the beginning, but it only reaches 45 % for the fraction of successful runs.

Algorithm	Control parameters
ASSA	(10.0, 0.8, 20, 1, 1.0, 5)
DE	(20, 0.5, 0.9)
ECTS	(10, 4, 10)
NA	(500, 100, 2)

In the control parameters for ASSA, T_0 is displayed instead of `t0fact`.

Goldstein-Price

All of the four optimisation algorithms give similar results for Goldstein-Price, as displayed in figure 7.4. Only ECTS, with its 95 % success fraction, does not succeed in all the runs. NA is the fastest of the algorithms. It should also be noted that the initial sample size for NA is 500 models.

Algorithm	Control parameters
ASSA	(20.0, 0.95, 1, 1, 3.0, 1)
DE	(20, 0.5, 0.9)
ECTS	(7, 20, 10)
NA	(500, 2, 2)

Shubert

The results for Shubert are displayed in figure 7.5. Here, ASSA and NA have very similar results, both reaching success fractions of 100 %. DE reaches 100 % as well, but it needs far more evaluations to do so. ECTS is fast, but it only reaches a success fraction of 85 %.

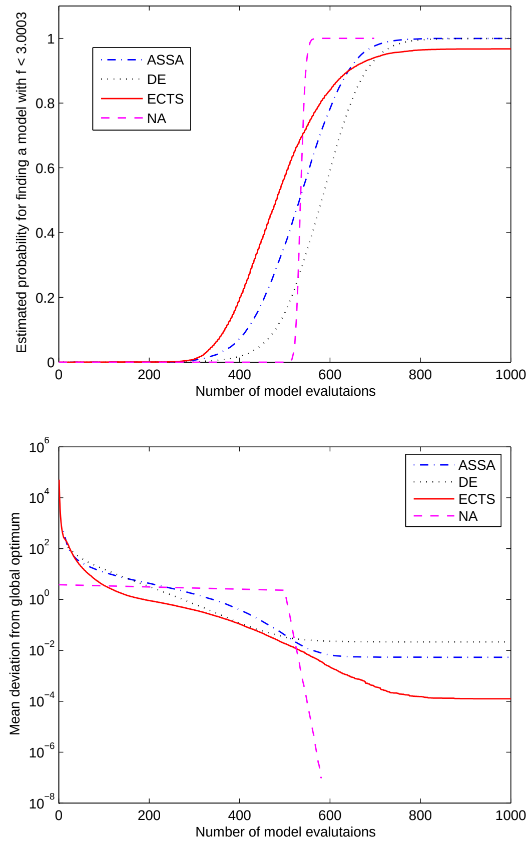


Figure 7.4: A comparison of optimisations of Goldstein-Price. Phases chosen are: ASSA: (20.0, 0.95, 1, 1, 3.0, 1), DE: (20, 0.5, 0.9), ECTS: (7, 20, 10) and NA: (500, 2, 2)

Algorithm	Control parameters
ASSA	(30.0, 0.9, 30, 1, 3.0, 4)
DE	(20, 0.5, 0.9)
ECTS	(10, 16, 20)
NA	(500, 100, 20)

7.2 Discussion

The performances of the four optimisation algorithms vary with the objective function. It does not seem possible to choose an algorithm that works well on all optimisation problems. In most cases in these comparisons, ASSA and

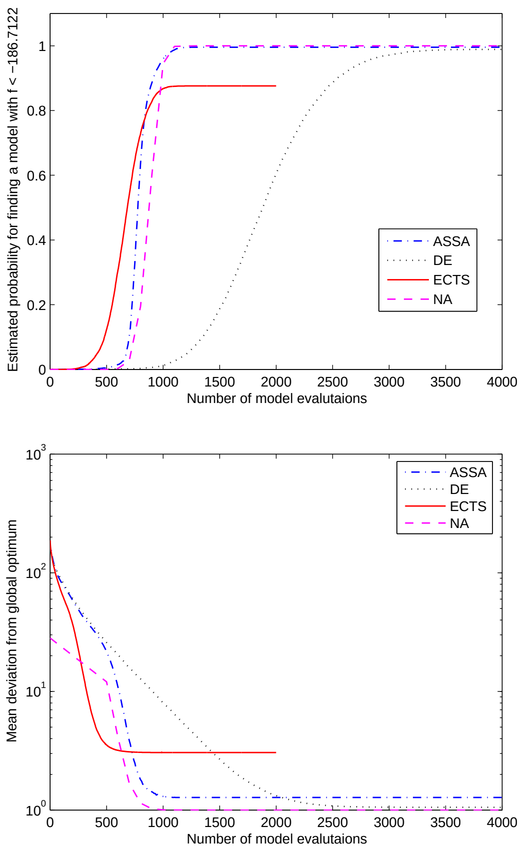


Figure 7.5: A comparison of optimisations of Shubert. Phases chosen are: ASSA: (30.0, 0.9, 30, 1, 3.0, 4), DE: (20, 0.5, 0.9), ECTS: (10, 16, 20) and NA: (500, 100, 20)

DE are the most reliable algorithms. However, ASSA has some problems to optimise Fallat-Dosso and DE is quite slow compared to the other algorithms when optimising Shubert. Generally, however, ASSA and DE seem to be good choices of algorithms for future optimisations of objective functions with unknown optima.

Concerning the results for ASSA when optimising the Fallat-Dosso test problem, it may be interesting to note another similar optimisation test. Fallat and Dosso used SSA, another algorithm combining Downhill Simplex and Simulated Annealing, to optimise the same test problem and obtained a success fraction of 93 % with at least twice or even three times as many evaluations as with ASSA [3]. However, Fallat and Dosso performed only 100 optimisation runs. Thus, the success probability estimate of 93 % has an uncertainty of perhaps ±3 % according to equation (2.2).

ECTS is the least reliable of the four algorithms, regardless of which of the five synthetic test problems is optimised. For most test problems, ECTS provides the fastest initial ascent of the success fraction curve. However, the curves flatten out well before reaching 100 %. Further research aiming at improvements of ECTS is desirable. Such improvements should probably be concentrated to the diversification stage, as discussed in chapter 6.

NA is originally designed for finding ensembles of models, rather than finding a single global optimum. Sambridge states that although NA is intended for this, it performed just as well as other global optimisation techniques in his tests [12]. However, NA does not perform as well as ASSA and DE in the optimisation examples studied in this chapter.

Sambridge also discusses a fusion between NA and other search methods, for example genetic algorithms [12]. This could be an interesting topic for further research in this field.

In Karlsson [2], a Simulated Annealing algorithm (SA) and a Genetic Algorithm (GA) are used to minimise Mexican hat. Since the testing procedure and the limit for success are similar to the ones used in this study, it is possible to compare the test results. The main difference between the two studies is that Karlsson uses $n_{\text{run}} = 100$, implying a larger maximum uncertainty $\sigma = 0.05$, while $n_{\text{run}} = 10000$ is used in this study (for NA, $n_{\text{run}} = 1000$).

During the first 1000 objective function evaluations, SA is more likely to find an objective function value below -0.99 than any of the four algorithms tested in this study. After that, DE and ASSA surpass SA, since SA reaches its maximum success fraction at 85%. GA reaches 85% after 2000 evaluations. This means that GA and SA perform worse than ASSA and DE but better than NA and ECTS.

Chapter 8

Acoustic Applications

Hanna Gothäll and Rune Westin

During the first world war, attempts were made to detect submarines by transmitting underwater sound pulses (called *pings*) and listening for the echoes. The British acronym for these detection devises was ASDIC (Anti-Submarine Detection Investigation Committee) and the US term, used in the second world war and thereafter, is SONAR (SOund, NAvigation and Ranging) [22]. The latter term is widely used today.

The time between a transmitted ping and the received echo provides the distance to the detected object, and the frequency shift in the response provides information about the movement direction of the object.

The submarine, on the other hand, uses countermeasures to improve its ability to avoid sonar detection. One such countermeasure is an anechoic coating on the outer hull of the submarine that reduces the sound power being reflected back to the sonar. There are several ways to design these coatings.

When avoiding detection, it is mainly the reflection coefficient of normally incident sound that is relevant since it is this sound that will be reflected back to a monostatic sonar (a sonar with the transmitter and the receiver close together. A bistatic sonar, however, can detect echoes from sound with other angles of incidence than 0°.).

This chapter deals with the optimisation of anechoic coatings. First, the geometry and material properties of a classic type of coatings are optimised. These coatings are made of rubber and they contain air-filled cavities.

Then, a study of the material in certain coatings is described briefly. The sound velocity and sound absorption profiles of these materials can also be optimised.

8.1 Alberich Anechoic Coatings

8.1.1 Background

Anechoic coatings have been in use since the second world war, when the German navy used rubber coatings with air-filled cavities to reduce the echoes from the hull of its submarines. The code name for these anechoic coatings, that cover the outer steel hull of the submarine (see figure 8.1), was "Alberich" [23].

The principal idea of Alberich anechoic coatings is that incident sound energy from an active sonar, wich enters the rubber coating, is scattered by the cavities and dissipated by anelastic attenuation in the rubber (see figure 8.2). The amount of sound energy reflected back to the sonar is thus reduced.

There are several ways to design anechoic

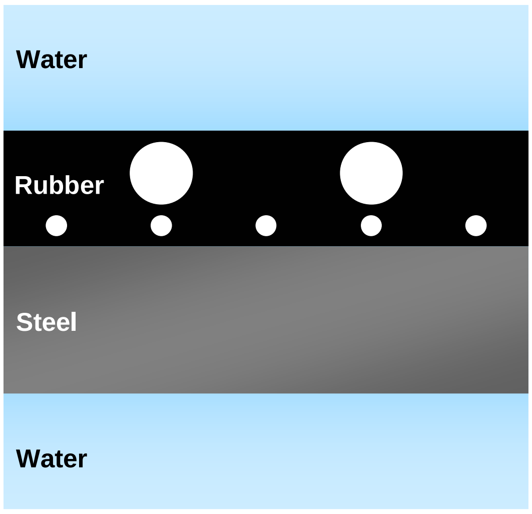


Figure 8.1: An illustration of an Alberich anechoic coating on the outer hull of a submarine.

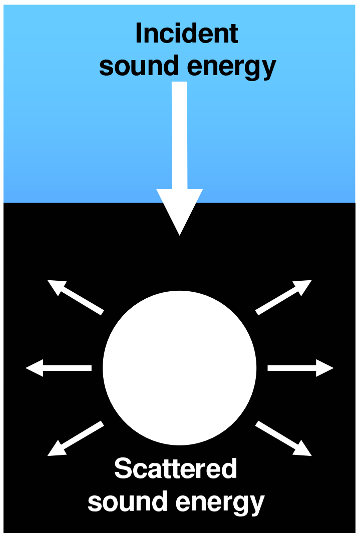


Figure 8.2: The function of the cavities in the rubber coatings. Normally incident sound energy is scattered and dissipated by anelastic attenuation.

coatings. The thickness can range from a few millimetres up to 100 mm. However, the use of cavities in the coatings reduces the thickness needed. The cavities can be placed in one or several layers and they can have equal or different sizes and shapes. An important question is: How should the coatings be designed to minimise the sound reflection coefficient?

8.1.2 Problem Description

In order to minimise the reflection coefficient of the anechoic coatings, an optimisation problem is formulated based on a theoretical model of the coatings.

Theoretical Model

Ivansson and Frenje Lund have described methods to calculate the reflection coefficient of certain kinds of Alberich anechoic coatings for different sound frequencies [24]. The coating to be considered here has two layers of spherical cavities distributed at equal distances from each other (see figure 8.1). In practice, coatings with cylindrical cavities are more common, but coatings with spherical cavities are easier to handle computationally.

Based on the method in [24], Ivansson has written a Fortran 77 program calculating the reflection coefficient of Alberich coatings for different frequencies. For the application considered in this report, the input parameters comprise three variables representing acoustic characteristics of the rubber, and seven variables representing geometrical characteristics of the rubber and the cavities.

The program returns the reflection coefficient of a coating described by the ten variables, i.e., the model **m**. The variables are:

1. Factor for distance between cavities
2. Shear wave velocity in the rubber (m/s)

3. Shear wave damping in the rubber (dB/λ)
4. Factor for the radii of the upper cavities
5. Factor for horizontal shift between the cavities in the two layers
6. Factor for the distance between the cavity layers
7. Factor for the radii of the lower cavities
8. Factor for extra rubber thickness
9. Factor for rubber thickness above the upper cavities
10. Longitudinal wave damping (dB/λ)

The boundaries of the search space are set to limit the optimisation to feasible coating designs. For example, a rubber material with the specified acoustic characteristics should be possible to manufacture.

The thickness of the outer steel hull is here set to six millimetres. The thickness of the rubber coating is required to be less than six millimetres. The radii of the cavities are between 0.25 and 2.5 millimetres, and the distances between the upper cavities range from about 2.5 to 12.5 cavity radii.

Two of the model variables, m_5 (horizontal shift) and m_{10} (longitudinal wave damping), are set to be constant, meaning that the upper limits of these variables are equal to the lower limits.

Objective Function Value

To be precise, the calculated reflection coefficient for a particular $\mathbf{m}$ is the quotient between the corresponding normally reflected sound power W_r and the normally incident sound power W_i. The reflection coefficient, in dB scale, is

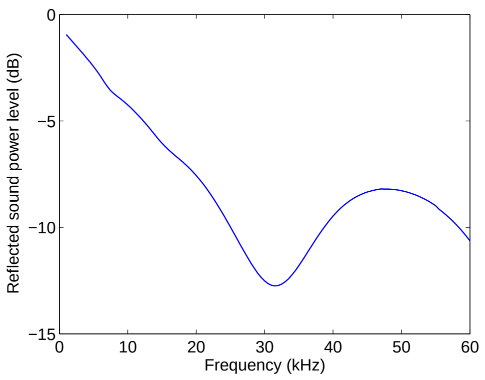


Figure 8.3: A sample reflection coefficient as a function of frequency.

$$L_W(f_s, \mathbf{m}) = 10 \cdot \log_{10} \left(\frac{W_r(f_s, \mathbf{m})}{W_i(f_s, \mathbf{m})} \right), \quad (8.1)$$

where f_s is the sound frequency (see example in figure 8.3). The objective function value to be minimised is defined as *the maximum reflected sound power level between 15 and 30 kHz.* This means that only the frequency band between 15 an 30 kHz is considered (see figure 8.4). Thus, the expression for $f(\mathbf{m})$ becomes

$$\begin{aligned} f(\mathbf{m}) = \max(L_W(f_s, \mathbf{m})), \\ 15 \text{ kHz} \leq f_s \leq 30 \text{ kHz} \end{aligned} \quad (8.2)$$

8.1.3 Results

Evaluating the reflection coefficient of a certain Alberich model is much more computationally cumbersome than evaluating the objective function value of any of the synthetic test functions described in chapter 2. On the computers available in the study, one objective function evaluation takes two to three seconds.

Also, the fact that the models consist of ten variables (eight actually, since two of the variables are fixed) indicates that the optimisation of the Alberich coating design requires more

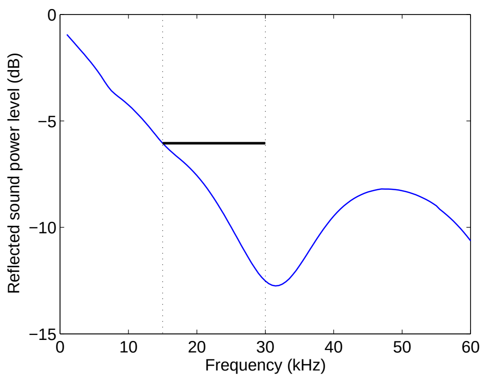


Figure 8.4: The horizontal black solid line shows the maximum reflected sound power level between 15 and 30 kHz. This is the objective function value to be minimised.

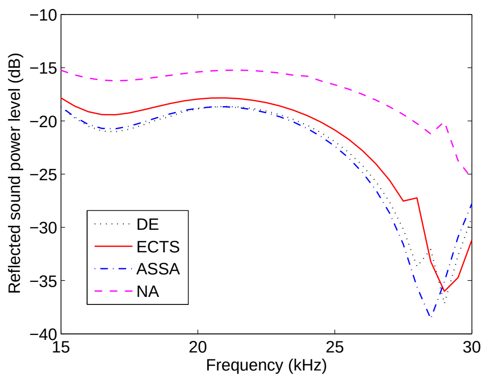


Figure 8.5: The reflectivities of the best designs found by the different optimisation algorithms. The reflectivities are shown in the considered frequency band (15-30 kHz).

evaluations than the optimisation of the synthetic test functions.

Therefore, exhaustive testing with different control parameters was not possible with the limited time available. The experience obtained about control parameters during the test function optimisations was used to select control parameter setups for the optimisation techniques.

The optimisation of the coating was performed up to five times with each technique. When necessary, the control parameters were tuned for the optimisations to give better results. The best reflection coefficient curves found by the different techniques are shown in figure 8.5. The control parameters used in the best optimisation runs are given for each algorithm.

ASSA

ASSA required 8500 evaluations to find the model with the reflection coefficient shown in figure 8.5. Based on all optimisations in the test, this is believed to be close to the global optimum. This was also similar to the best results in all performed ASSA optimisations, although sometimes, over 40000 evaluations were needed.

The control parameters used were (T_0, β, S, `nPert`, s , `nPertfactor`) = (20.0, 0.995, 1, 1, 3.0 , 3).

DE

The best result by DE was close to the best ASSA result. However, DE normally used approximately 40000 evaluations. The results of DE were consistent among the performed optimisations.

The control parameters used were (NP, F, CR) = (300, 0.7 ,0.9).

ECTS

ECTS was the fastest of the techniques, using only about 4000 evaluations. However, it never found the global optimum, and the results were not consistent among the different runs.

The control parameters used were (N_t, Max_p, Max_b) = (100, 80, 50).

NA

NA used about 20000 evaluations to find the model with the reflection coefficient shown in figure 8.5. This was a typical result for NA.

The control parameters used were (`nsamplei`, n_s, n_r) = (10000, 15, 15).

8.2 Material Profile Design

In the example with the Alberich anechoic coatings above, the rubber material between the cavities is homogenous. This section deals very briefly with a coating where the sound velocities and dampings of the material vary with the depth of the coating.

Such coatings have been considered by Björkert et al [25]. The sound velocities and dampings of longitudinal as well as transverse waves vary with the depth. The acoustic absorber adheres to the outer steel hull of the submarine.

In order to find the optimum material profile design for the acoustic absorber, the authors of [25] used a trust region method for nonlinear least-squares minimisation problems.

8.2.1 Algorithm Tests

It was desired to test if ASSA, DE, ECTS and NA could reproduce the results of the original optimisation in [25]. The four optimisation techniques were used on the material profile design problem (an optimisation problem of 24 variables).

Without going into details, it was found that all four techniques could match, and in some cases slightly improve, the results given in [25].

8.3 Discussion

In its best optimisation run on the Alberich anechoic coatings, the performance of ASSA

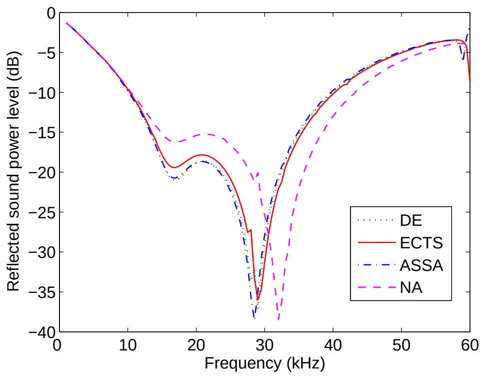


Figure 8.6: The reflectivities from figure 8.5 shown from one to 60 kHz.

was unmatched by the other optimisation techniques when considering both speed and obtained objective function value. However, this run was exceptional and in most cases, ASSA and DE gave similar results when optimising the coating design. One conclusion is that the results may depend much on the choice of control parameter values and, to some extent, luck. If the algorithm happens to start in a fortunate place in the search domain, it is more likely to find the global optimum fast.

ECTS was as usual very fast but not very reliable. If the diversification stage of ECTS encounters a good enough promising region, from which to intensify the search later on, ECTS will presumably be the fastest technique to find the global optimum. However, the required promising region appears to be very small compared to the search domain, and it is not likely that the diversification stage will be successful enough.

NA did not succeed in finding the optimum coating design. In its present version, the algorithm appears to be too greedy, with a too poor ability to climb out of local minima.

It should be pointed out that the assumed global optimum of the coating design is resided at an edge of the search domain. Both the factor for the distance between cavities (m_1) and the shear wave damping (m_3) are at the upper limits of their allowed ranges. Whether this affects the optimisation or not is unclear. However, a knowledge that some optimal model variable is at an edge of the domain could be used to reduce the number of dimensions in the optimisation problem, since variable values could be fixed. This would enhance the performance of all algorithms.

Chapter 9

A Bayesian Inversion Problem

Hanna Gothäll and Rune Westin

Inverse problems are problems where certain values or data are known and one wishes to find the model associated with these data. For example, an inverse problem arises when a signal has been received and one wishes to locate the source of the signal based on available theories of signal propagation. Inversion is used to fit variables to data.

More specifically, suppose that $\mathbf{d}$ is the known data, where $\mathbf{d}$ can be any set of data. Furthermore, let $\mathbf{m}_{\text{orig}}$ be the unknown original model that $\mathbf{d}$ is associated with. If $\mathbf{g}(\mathbf{m})$ is a vector-valued function based on a theory on how data are associated with models, $\mathbf{d} \approx \mathbf{g}(\mathbf{m}_{\text{orig}})$. To compute $\mathbf{g}(\mathbf{m})$ for a given $\mathbf{m}$ is called to solve the *forward problem.*

An objective function $f(\mathbf{m})$ can be defined according to

$$f(\mathbf{m}) = ||\mathbf{g}(\mathbf{m}) - \mathbf{d}||, \tag{9.1}$$

where $||\ldots||$ denotes a norm chosen by the user. This objective function, for which the value is always equal to or greater than zero, signifies the data *misfit* corresponding to the model $\mathbf{m}$. Since $\mathbf{g}(\mathbf{m}_{\text{orig}}) \approx \mathbf{d}$, it can be assumed that $f(\mathbf{m}_{\text{orig}}) \approx 0$, providing an opportunity to find the unknown original model $\mathbf{m}_{\text{orig}}$ by minimising $f(\mathbf{m})$.

Actually, in inverse problems the objective function value f is often called *misfit* to emphasise its relation to how well a model fits the data. It will also be the term used in this report when dealing with inverse problems.

This chapter deals with the inversion of a synthetic reflection coefficient from the Alberich anechoic coatings described in chapter 8.

9.1 A Synthetic Inverse Problem on Alberich Anechoic Coatings

In addition to ordinary optimisation, it was desired to test the four optimisation techniques on an inversion problem. For this, an inversion problem was designed based on the Alberich anechoic coatings (see chapter 8), since a computer program for calculating the coating reflection coefficient associated with a model was available.

9.1.1 Description of the Problem

It was decided to simulate the measurement of a certain reflection coefficient and try to find the model variables of the corresponding Alberich anechoic coating. The computer program for calculating the reflection coefficient, associated with a certain coating, provides the function $\mathbf{g}(\mathbf{m})$, where the model $\mathbf{m}$ describes the coating design.

To simulate a measurement, a feasible reflection coefficient was needed. An original model $\mathbf{m}_{\text{orig}}$ was chosen and the associated reflection coefficient $\mathbf{g}(\mathbf{m}_{\text{orig}})$ was calculated for $N_f = 16$ uniformly spaced frequencies between 15 and 30 kHz. In order to make the "measured" reflection coefficient a bit more realistic, a vector $\mathbf{e}$ of Gaussian noise was added to the reflection coefficient to simulate measurement noise. Thus, the original data were summarised in a vector given by $\mathbf{d} = \mathbf{g}(\mathbf{m}_{\text{orig}}) + \mathbf{e}$. The Gaussian noise component variables were identically distributed and independent.

The problem is to find the "unknown" original model $\mathbf{m}_{\text{orig}}$. Given $\mathbf{d}$ and $\mathbf{g}(\mathbf{m})$, $\mathbf{m}_{\text{orig}}$ can be estimated by minimising $f(\mathbf{m}) = ||\mathbf{g}(\mathbf{m}) - \mathbf{d}||^2$. The norm is here defined as

$$||\mathbf{a}||^2 = \sum_{i=1}^{N_f} a_i^2, \tag{9.2}$$

where N_f is the number of frequency components for the reflection coefficient. Therefore, the inversion problem can be formulated as a least-squares minimisation of the residual vector $\mathbf{g}(\mathbf{m}) - \mathbf{d}$.

The simulated Gaussian measurement noise had a standard deviation of $3 \cdot 10^{-3}$, giving the original model $\mathbf{m}_{\text{orig}}$ a misfit of

$$\begin{aligned} f(\mathbf{m}_{\text{orig}}) =& ||\mathbf{g}(\mathbf{m}_{\text{orig}}) - \mathbf{d}||^2 = \\ =& ||\mathbf{g}(\mathbf{m}_{\text{orig}}) - \mathbf{g}(\mathbf{m}_{\text{orig}}) - \mathbf{e}||^2 = \\ =& ||\mathbf{e}||^2 \approx 1.44 \cdot 10^{-4} \end{aligned} \tag{9.3}$$

for $N_f = 16$ (see equation (9.2)). This should be considered the lowest misfit of interest, since misfits below $f(\mathbf{m}_{\text{orig}})$ are likely to be more adapted to the measurement noise than to the original reflection coefficient.

9.1.2 Least-Squares Minimisation Results

The four optimisation techniques were used on the inversion problem and stopped when they reached the misfit of the original model. The numbers of evaluations needed for the different techniques to reach $f(\mathbf{m}_{\text{orig}})$ are shown in the table below. ECTS was the fastest of the four, DE and ASSA used approximately twice as many evaluations as ECTS. NA used more than three times as many evaluations as ECTS making it the slowest of the techniques.

Algorithm	Number of evaluations
ECTS	5 500
DE	9 600
ASSA	10 700
NA	17 700

The control parameters giving these results for the different algorithms are:
ASSA: (T_0, β, S, `nPert`, s, `nPertfactor`) = (30.0, 0.975, 10, 5, 1.0, 3)
DE: (NP, F, CR) = (80, 0.8, 0.9)
ECTS: (N_t, Max_p, Max_b) = (100, 100, 50)
NA: (`nsamplei`, n_s, n_r) = (5000, 150, 100).

9.2 Neighbourhood Algorithm – Bayes

Until now, the inversion problem has been treated as a particular optimisation problem, involving the search of a "best" model. However, it is not certain that this model is indeed $\mathbf{m}_{\text{orig}}$. Suppose, for example, that there are many models in the search domain associated with similar reflectivities. The question is: What is the degree of confidence that the found model agrees well with the original model?

In order to answer this question, knowledge of the accuracy of the data is needed, as is some prior idea (before taking the data into account) of a probability distribution in the search domain. A more general question is: What is the probability density for the estimated location in the search domain of the original model?

Prior Probability Density

Sometimes, there is a prior idea of the probability density. This *prior probability density*, denoted $\rho(\mathbf{m})$, may arise from some former knowledge about the problem. For example, suppose that the problem is to estimate the length of a person, in a known population, from some data. It seems reasonable to assign a normally distributed prior probability density.

In many cases, there is no prior preference concerning different locations in search space. The prior probability density is typically chosen to be uniform within the search domain under such circumstances ($\rho(\mathbf{m}) = \text{constant}$).

Posterior Probability Density

If the misfit $f(\mathbf{m}) = ||\mathbf{g}(\mathbf{m}) - \mathbf{d}||^2$ is known for all models and if the noise of the data $\mathbf{d}$ (as defined in section 9.1.1) is Gaussian with a standard deviation of σ, the natural *posterior probability density (PPD)*, denoted $P(\mathbf{m})$, fulfills

$$P(\mathbf{m}) \sim \rho(\mathbf{m}) \cdot e^{-\frac{f(\mathbf{m})}{2\sigma^2}} \tag{9.4}$$

according to Bayes' rule [26], [27].

Apparently, the right-hand side of equation (9.4) can be evaluated for each particular model $\mathbf{m}$ in the search domain. For example, the relative posterior probability for a number of specified models can directly be determined. Absolute PPD values are more difficult to obtain, however, since the proportionality constant inherent in equation (9.4) is unknown. This proportionality constant is needed to achieve $\int P(\mathbf{m}) \cdot d\mathbf{m} = 1$. Important questions are how to resolve this proportionality problem and how to determine the regions in the search domain with high PPD values.

9.2.1 The Search Ensemble

When a least-squares solution to an inverse problem has been obtained, as in section 9.1.2, much more information than the single best model found is actually available. A whole ensemble of calculated models has been collected during the search! The question is if the information in this search ensemble can be used to gain information about the PPD.

An immediate idea is to plot ensemble marginal distributions for different search domain variables. Figure 9.1 shows marginal distributions for the search ensemble from the least-squares minimisation done by DE (section 9.1.2). Four model variables are chosen here: the factor for the distance between cavities, the rubber shear wave velocity, the rubber shear wave damping and the factor for cavity radius. The original parameters are indicated with vertical dotted lines.

It is not certain, however, that these distributions give a truthful indication of the uncertainty in the estimation of the variables of the original model. The sampling distribution of the DE search algorithm is unknown, and it could deviate significantly from the PPD (biased sampling). The search algorithm could, for example, favour a domain with low misfit values too much and sample this domain unrepresentatively much.

9.2.2 Neighbourhood Approximation

The search ensemble could be used as a guide for resampling the search domain. The idea is to substitute the true PPD with an approximate PPD, and resample the search domain accordingly. The resampling generates a new

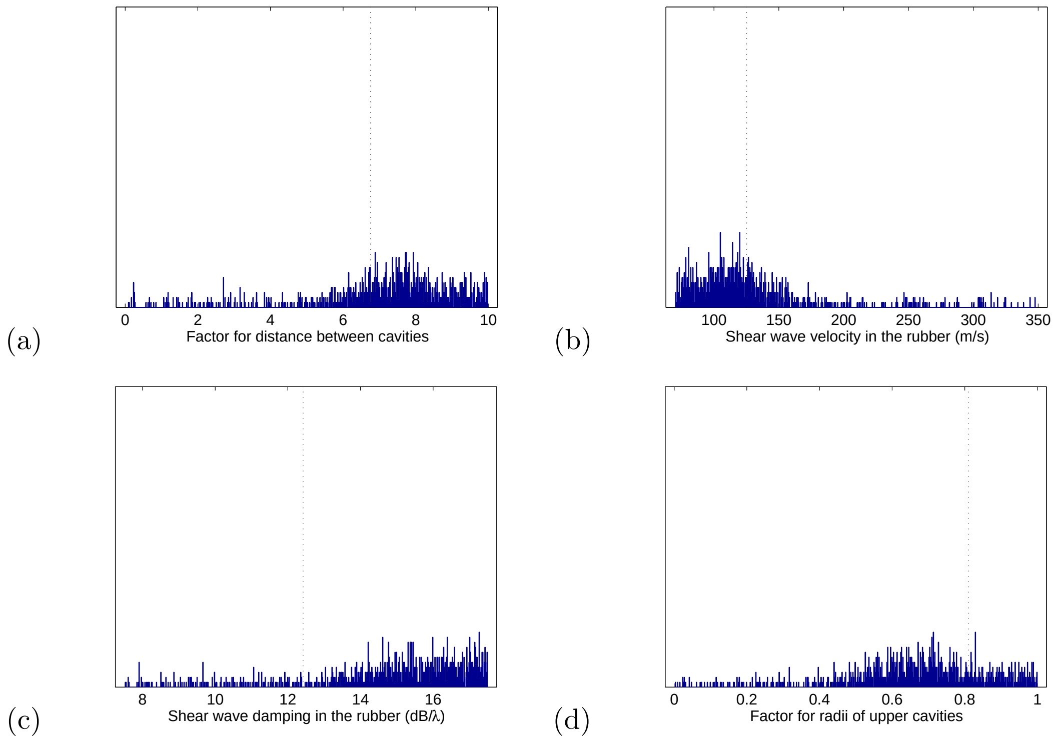


Figure 9.1: Marginal distributions for the search ensemble from the least-squares minimisation with DE. The following four model variables are displayed: (a) the factor for the distance between cavities, (b) the rubber shear wave velocity, (c) the rubber shear wave damping and (d) the factor for the cavity radius. The original parameters are indicated with vertical dotted lines.

ensemble distributed according to the approximate PPD.

The *neighbourhood approximation to the PPD* is obtained by setting the model misfit to a constant inside each Voronoi cell. It is denoted $P_{\mathrm{NA}}(\mathbf{m})$, and represents all the information gained from the input search ensemble. $P_{\mathrm{NA}}(\mathbf{m})$ is here chosen to guide the resampling.

A resampling algorithm exists, see section 9.2.3, for which the sampling distribution tends to the target distribution, $P_{\mathrm{NA}}(\mathbf{m})$, after many iterations [26]. An advantage is that the forward problem need not be solved again during the resampling.

When an ensemble distributed according to the approximate PPD has been obtained, the moments of the distribution can be estimated by simple averaging. Hence, from the resampled ensemble, correlations and uncertainties can be estimated, among other things.

9.2.3 Description of Neighbourhood Algorithm – Bayes

The algorithm used to do this resampling is called Neighbourhood Algorithm – Bayes (NAB). A short description of the method and the theory behind it is given here. For a more thorough exposition, see Sambridge [26].

Since only an ensemble of models generated during a search is needed, NAB can be used for any search method which has been used to solve the inverse problem.

In contrast to NA, the Voronoi cells are unchanged during a calculation with NAB. The new models created are only collected to shape a distribution. Hence, the Voronoi cell structure is never changed.

NAB uses importance sampling, hence, it does not waste time in domains with high misfit. Instead, NAB concentrates the work to the domains with low misfit. The importance sampling of the approximate PPD is done with a technique known as Gibbs sampler, which is effective in the Voronoi cell context. The size and structure of a Voronoi cell get more and more complex with increasing dimensionality of the problem. In higher dimensions, the size and structure are extremely difficult to calculate. The Gibbs sampler walks in only one variable direction at the time. Along a line, the intersections with the Voronoi cells are easy to calculate. Hence, the dimensionality of the problem causes no particular difficulties.

NAB performs a number of random walks in the search domain. The initial cell for the first random walk, for example cell i, is determined by a control parameter. In NAB, in contrast to NA, each random walk steps outside the initial Voronoi cell in order to resample the ensemble (see figure 9.2). Axes directions for the random walk are cycled through until all axes have been considered. The probability of stepping into a cell is determined by the product of its PPD value and the width of its intersection with the current axis. After each cycle of steps, a new $\mathbf{m}$ is typically collected to the ensemble. The cycles are done a number of times until the prescribed number of steps in the random walk have been done. The second walk starts in cell $i + 1$, and so on until all random walks have been made.

An alternative to NAB is the Metropolis Algorithm (cf. Simulated Annealing at $T = 1$), which solves the inverse problem with a vast number of evaluations of $\mathbf{g}(\mathbf{m})$. A faster variant of the Metropolis Algorithm is Dosso's Fast Gibbs Sampler (FGS) [27] .

9.2.4 Control Parameters

The control parameters and some suggested values are specified here. More information can be found in [14].

- Number of random walks (100)
- Number of steps per random walk (1000)

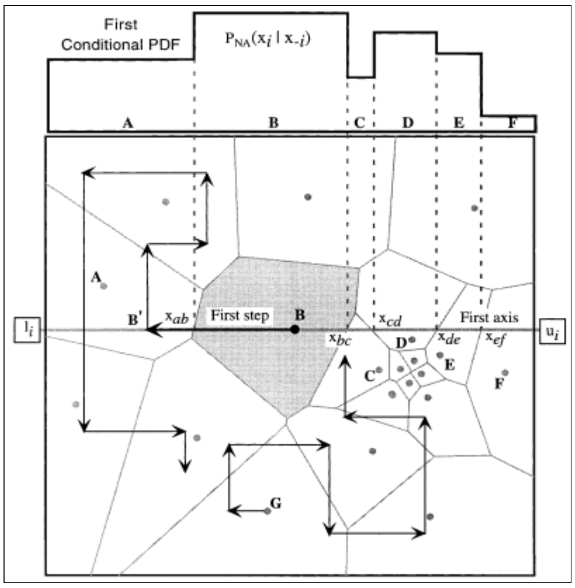


Figure 9.2: Two independent random walks guided by the neighbourhood approximation of the PPD. The Gibbs sampler is used. For the first step (x-direction) of the walk starting at cell B (shaded) the shape of the conditional, $P_{\mathrm{NA}}(x_i|x_{-i})$, is shown above the figure. After many steps, the density distribution of the random walk will asymptotically tend to the approximate PPD, $P_{\mathrm{NA}}(\mathbf{x})$. (Picture and caption after [26].)

- Starting Voronoi cell of first random walk (1) (1=lowest misfit, 2=second lowest misfit,...)
- Number of direction axes cycles to make before collecting a model to the new ensemble (1)
- Number of steps before refreshing internal tables (avoids error build up) (2000)
- Several options for numerical integration (see [14] for more information)

9.2.5 Enhancement

When running the NAB program, it terminated before it had completed all random walks in some runs. This happened when the program tried to randomly choose a value along an axis according to a certain distribution. This is done in NAB with a rejection method in the subroutine `NA_randev`. A number is randomly chosen (uniformly distributed) within the search range of the variable in question. The number is accepted at once if the corresponding cell has the highest PPD value along the axis. If this is not the case, the number is accepted with a probability based on the PPD value of the cell. If the number is rejected, the method tries all over again.

The modification to the subroutine is called `NA_randev_alt` [1] and it is an additional part of the subroutine `NA_randev` that is activated in the case of too many rejections. It is a direct distribution function method, which temporarily rescales the lengths of the cells along the axis. The length of a cell is, after the rescaling, its width times its PPD value. A random number is uniformly chosen. It is used to determine the next model for the random walk, along the axis.

9.2.6 Results

Marginal distributions after the resampling are shown in figure 9.3. The model variables are the same as in section 9.1. The original model variables are shown as vertical dotted lines here as well.

Clearly, changes can be seen compared to the marginal distributions from the original DE least-squares minimisation (see figure 9.1). For the factor for the distance between cavities (panel (d)), a drastic change has appeared: the distribution is uniform after the resampling, although it looked quite different before the

[1]the code is written by Sven Ivansson

resampling. On the other hand, the rubber shear wave damping (panel (c)) is correctly and accurately determined, although another impression was obtained from the initial search ensemble. In the distribution for the rubber shear wave velocity (panel (b)), an ambiguity can be seen with essentially two possible values. For the factor for the distance between cavities (panel (a)), NAB seems to have concentrated the distribution to the region which looked most important before the resampling.

In figure 9.4, two different two-dimensional distributions are displayed. The original model variables are shown with white circles. This kind of plots can visualise correlations between the variables. Indeed, a diagonal appearance can be seen in both panels. The plots also indicate uncertainties in the model variable estimates, by the spread in the corresponding directions.

9.2.7 Discussion

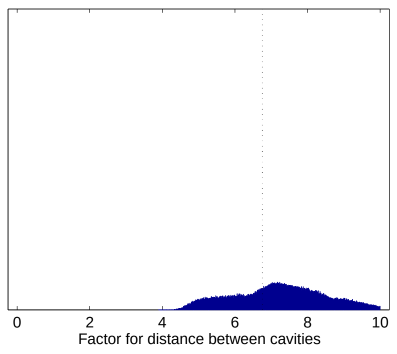
NAB has shown to be useful in connection with inverse problems for evaluating the uncertainties of the estimates of the model variables. Starting from a search ensemble created by any global optimisation method, NAB does not require any further evaluations of the forward problem.

A drawback of NAB is that the regions with low misfit in the search domain must be well sampled. Otherwise, the neighbourhood approximation to the PPD will not be representative. This means that the ensembles from certain algorithms (for example ECTS, which is not always able to find good promising regions) are not suitable.

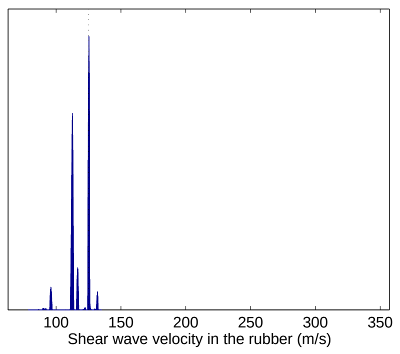
The Fast Gibbs Sampler (FGS) algorithm by Dosso [27] aims at sampling from the true PPD, but a vast number of forward problem solutions are needed.

The NAB computer program contains more features than discussed in this chapter. For example, the output files contain distribution moments and there is an indicator variable telling whether or not enough models have been collected. The NAB program package is well suited and prepared for parallel computing.

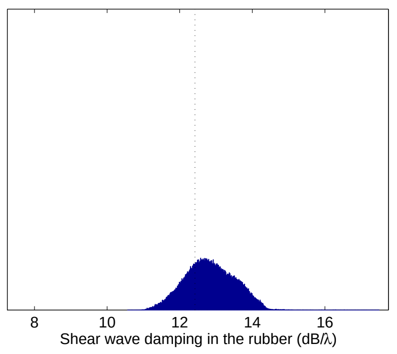
(a)



(b)



(c)



(d)

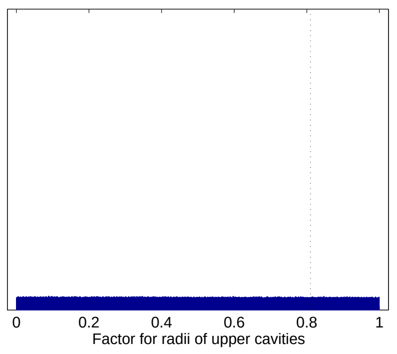
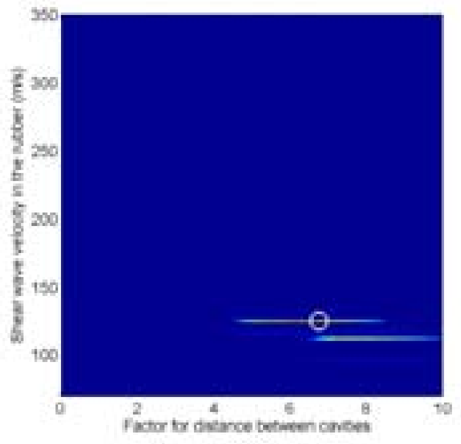


Figure 9.3: Marginal distributions for the ensemble collected with NAB. The following four model variables are displayed (the same as before the resampling): (a) the factor for the distance between cavities, (b) the rubber shear wave velocity, (c) the rubber shear wave damping and (d) the factor for the cavity radius. The original model variables are indicated with vertical dotted lines.



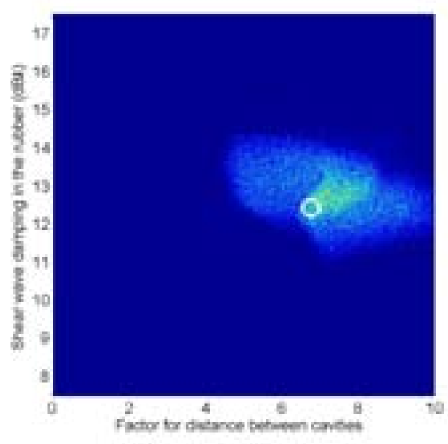


Figure 9.4: Two-dimensional distributions for the ensemble collected with NAB. The following model variables are displayed: (a) the rubber shear wave velocity versus the factor for the distance between cavities and (b) the rubber shear wave damping versus the factor for the distance between cavities. The original model variables are indicated with white circles.

Chapter 10

Conclusions

Hanna Gothäll and Rune Westin

As discussed in the previous chapters, the most reliable of the four optimisation algorithms seem to be ASSA and DE.

10.1 ASSA

ASSA is a fast optimisation technique, which is also reliable (particularly after the improvements, see sections 3.4.1 and 3.4.2). The basic ideas are easy to understand. The test problem Fallat-Dosso proved difficult, but in all other cases ASSA provided good results quickly.

An advantage of ASSA is that it is a combination of a global and a local optimisation technique. It is a good global optimiser, and when the temperature gets low, it is a good local optimiser as well.

There are many control parameters, which makes it a bit difficult to adjust the values. Still, the suggested values have, in the tests in this study, proved to be good (at least as something to start from).

10.2 DE

DE has proved to be a very reliable technique for a wide range of optimisation problems. A few exceptional test problems have been found, however, for which DE is comparatively slow. The algorithm is easy to comprehend and implement, and the control parameters are few. DE is also well suited for parallel computing.

10.3 NA

NA is an algorithm which is very easy to understand. The algorithm is not that easy to implement, but the NA program package is available (see [14]) and is well suited and prepared for parallel computing. Another advantage is that the number of control parameters is so small.

A drawback is that the algorithm is too greedy: in the test examples, NA does not reach a 100% success fraction as often as ASSA and DE do. An idea for improvement could be to allow a cell to be chosen for a random walk with a certain probability, even though its objective function value is higher than the n_r lowest objective function values. This probability should preferably decrease with time.

10.4 ECTS

ECTS appears to be a good local optimiser with global abilities. However, even though the local abilities of ECTS may be better than for DE and NA, they are not as good as for dedicated local algorithms. At the same time, ECTS is outperformed in global optimisation by the other optimisation techniques in this study.

The foremost advantage of ECTS is its speed. In optimisation runs where it finds a model close enough to the global optimum, ECTS is fast compared to most other algorithms.

In most problems, there are probably better choices than ECTS. It could be preferable to use a global algorithm, with better global abilities than ECTS, with a dedicated local optimiser added as a final step in the optimisation. An interesting research problem would be to try to improve the diversification stage of ECTS.

10.5 NAB

NAB has proved to be useful in connection with Bayesian inverse problems to assess uncertainties in the model variable estimates.

Advantages of NAB are that it does not require any further evaluations of the forward problem, and that it can be applied to an ensemble generated by any global optimisation technique.

The NAB program package is well suited and prepared for parallel computing.

A drawback of NAB is that the regions with low misfit in the search domain must have been well sampled. Otherwise, the neighbourhood approximation to the PPD will not be representative.

Bibliography

[1] Rachid Chelouah and Patrick Siarry. Tabu search applied to global optimization. *European Journal of Operational Research*, 123(2):256–270, 2000.

[2] Mattias Karlsson. Comparison of local and global search algorithms for inverse electromagnetic problems. Technical report, FOI - Swedish Defence Research Agency, Systems Technology, March 2003.

[3] Mark R. Fallat and Stan E. Dosso. Geoacoustic inversion via local, global, and hybrid algorithms. *The Journal of the Acoustical Society of America*, 105(6):3219–3230, 1999.

[4] J.A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7(4):308–313, 1965.

[5] Gregorz Kaczmarczyk. Downhill simplex method for many (∼20) dimensions. `http://paula.univ.gda.pl/~dokgrk/simplex.html`.

[6] Stan E. Dosso, Michael J. Wilmut, and Anna-Liesa S. Lapinski. An adaptive-hybrid algorithm for geoacoustic inversion. *IEEE Journal of Oceanic Engineering*, 26(3):324–336, 2001.

[7] Stan E. Dosso, Ronald T. Kessel, and Mark R. Fallat. Hybrid matched-field inversion for geoacoustic properties and uncertainties. In A. Alippi and G. B. Cannelli, editors, *Proceedings of the Fourth European Conference on Underwater Acoustics*, volume 1, pages 313–318, September 1998.

[8] Abdel-Rahman Hedar and Masao Fukushima. Hybrid simulated annealing and direct search method for nonlinear unconstrained global optimization. *Optimization Methods and Software*, 17:891–912, 2002.

[9] Rainer Storn and Kenneth Price. Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical report, ICSI, March 1995.

[10] Rainer Storn and Kenneth Price. Minimizing the real functions of the icec'96 contest by differential evolution. In *IEEE Conference on Evolutionary Computation, Nagoya*, pages 842 – 844, May 1996.

[11] Rainer Storn et al. Differential evolution homepage. `http://www.icsi.berkeley.edu/~storn/code.html`.

[12] Malcolm Sambridge. Geophysical inversion with a neighbourhood algorithm - 1. searching a parameter space. *Geophysical Journal International*, 138(2):479–494, 1999.

[13] Joseph S. Resovsky and Jeannot Trampert. Reliable mantle density error bars: an application of the neighbourhood algorithm to normal-mode and surface wave

data. *Geophysical Journal International*, 150(3):665–672, 2002.

[14] Malcolm Sambridge. Neighbourhood algorithm homepage. `http://rses.anu.edu.au/~malcolm/na/na.html`.

[15] Malcolm Sambridge. Finding acceptable models in nonlinear inverse problems using a neighbourhood algorithm. *Inverse Problems, Institute of Physical Publishing*, 17(3):387–403, 2001.

[16] Malcolm Sambridge. Nonlinear inversion by direct search using the neighbourhood algorithm. *International Handbook of Earthquake and Engineering Seismology*, 81(B):1635–1637, 2003.

[17] Rikke Vinther and Klaus Mosegaard. Seismic inversion through tabu search. *Geophysical Prospecting*, 44(4):555–570, 1996.

[18] Zoi-Heleni Michalopoulou and Urmi Ghosh-Dastidar. Tabu for matched-field source localization and geoacoustic inversion. *The Journal of the Acoustical Society of America*, 115(1):135–145, 2004.

[19] Zoi-Heleni Michalopoulou. Optimizing matched-field inversion using tabu search. In Dick G. Simons, editor, *Proceedings of the Seventh European Conference on Underwater Acoustics*, volume 2, pages 653–658, July 2004.

[20] Patrick Siarry and Gérard Berthiau. Fitting of tabu search to optimize functions of continuous variables. *International Journal for Numerical Methods in Enginering*, 40(13):2449–2457, 1997.

[21] Y.S. Teh and G.P. Rangaiah. Tabu search for global optimization of countinuous functions with application to phase equilibrium calculations. *Computers and Chemical Engineering*, 27(11):1665–1679, 2003.

[22] About Invetors. The history of sonar. `http://inventors.about.com/library/inventors/blsonar.htm`.

[23] Stan Zimmerman. *Submarine Technology for the 21st Century*. Trafford Publishing, second edition, 2000.

[24] Sven Ivansson and Lena Frenje Lund. Modelling of echo reduction by alberich anechoic coatings. Technical report, FOI - Swedish Defence Research Agency, Systems Technology, December 2003.

[25] Stefan Björkert, Ilkka Karasalo, Steven Savage, Örjan Staaf, and Sören Svensson. Novel technology for hydroacoustic signature management. Technical report, FOI - Swedish Defence Research Agency, Systems Technology, September 2003.

[26] Malcolm Sambridge. Geophysical inversion with a neighbourhood algorithm - 2. appraising the ensemble. *Geophysical Journal International*, 138(3):727–746, 1999.

[27] Stan E. Dosso. Quantifying uncertainty in geoacoustic inversion 1. a fast gibbs sampler approach. *The Journal of the Acoustical Society of America*, 111(1):129–142, 2002.