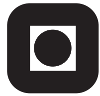

NTNU – Trondheim
Norwegian University of
Science and Technology

Study of Optimization Algorithms for Underwater Acoustic Applications

Niklas Saxlund Skyberg

Master of Science in Electronics
Submission date: June 2013
Supervisor: Hefeng Dong, IET

Norwegian University of Science and Technology
Department of Electronics and Telecommunications

PROBLEM DESCRIPTION

Different optimization algorithms have been applied to underwater acoustic applications, such as simulated annealing (SA), adaptive simplex simulated annealing (ASSA), genetic algorithms (GA), particle swarm (AS) and differential evolution (DE). All of these algorithms search the global minimum in the search space. However, the convergence speeds are different. In this project, metaheuristic optimization algorithms will be studied and adapted to an application in underwater acoustics: estimation of seismic velocities of upper oceanic crust from ocean bottom reflection loss data. The acoustic parameters in the sediment layer and the oceanic crust include compressional and shear wave velocities, their attenuations, densities in the two layers respectively and the thickness of the sediment layer. These will be the estimated parameters. The performances of the metaheuristic optimization algorithms are assessed in terms of accuracy and computational cost in solving this underwater acoustic problem.

NORWEGIAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Abstract

Department of Electronics and Telecommunications

Master of Science in Electronics

Study of Optimization Algorithms for Underwater Acoustic Applications

by Niklas Saxlund Skyberg

This thesis introduces the algorithms harmony search (HS) and artificial bee colony (ABC) to the field of geoacoustic inversion. HS mimics jazz musicians search of an optimal harmony through improvisation, while ABC is inspired by bee's search for food sources with a high amount of nectar. These global optimizers have been combined with downhill simplex (DS), a local optimizer, in order to create hybrid optimization algorithms. These hybrids have further been compared with a hybrid version of differential evolution (DE) by testing them on a problem of geoacoustic inversion. The goal was to see if these relatively recent algorithms could outperform the well known evolutionary algorithm. The hybrid based on HS is shown to have better performance than the DE-based hybrid, both in terms of computational cost and accuracy.

Norges teknisk-naturvitenskapelige universitet

Sammendrag

Institutt for elektronikk og telekommunikasjon

Sivilingeniør i elektronikk

Studie av optimiseringsalgoritmer for applikasjoner i undervannsakustikk

av Niklas Saxlund Skyberg

I denne masteroppgaven introduseres optimeringsalgoritmene harmony search (HS) og artificial bee colony (ABC) til bruk i geoakustisk inversjon. HS imiterer jazz musikeres søk etter en optimal harmoni gjennom improvisasjon, mens ABC er inspirert av biers søk etter matkilder med høyt innhold av nektar. Ved å kombinere HS og ABC med den lokale optimeringsalgoritmen downhill simplex (DS), har hybridutgaver av optimeringsalgoritmene blitt laget. Disse hybridene har så blitt sammenliknet med en tilsvarende hybridutgave av differential evolution (DE), ved å la dem løse et optimeringsproblem bestående av geoakustisk inversjon. DE har ved tidligere anledninger vist seg å være svært effektiv på liknende problemer, og er derfor en naturlig algoritme å sammenlikne med. Det vises videre at hybridutgaven av HS har bedre ytelse enn hybridutgaven av DE, både i form av lavere beregningskostnad og større nøyaktighet.

Preface

This thesis has been written at the Norwegian University of Science and Technology, the spring of 2013. The thesis counts for 30 academic points, and is the final part of a Master Degree in Electronics. I would like to thank the project supervisor, Hefeng Dong for essential help. I would also like to thank Marwan M. Fuad for introducing me to metaheuristic optimization during the project assignment last semester.

Niklas Saxlund Skyberg
NTNU, Trondheim
June, 2013

ABBREVIATIONS

ABC	**A**rtificial **B**ee **C**olony
ASABC	**A**daptive **S**implex **A**rtificial **B**ee **C**olony
ASDE	**A**daptive **S**implex **D**ifferential **E**volution
ASGA	**A**daptive **S**implex **G**enetic **A**lgorithm
ASHS	**A**daptive **S**implex **H**armony **S**earch
ASSA	**A**daptive **S**implex **S**imulated **A**nnealing
DE	**D**ifferential **E**volution
DS	**D**ownhilll **S**implex
GA	**G**enetic **A**lgorithm
HS	**H**armony **S**earch
SA	**S**imulated **A**nnealing

SYMBOLS

V_{p1}	Sediment compressional wave velocity (m/s)	
V_{p2}	Basalt compressional wave velocity (m/s)	
V_{s1}	Sediment shear wave velocity (m/s)	
V_{s1}	Basalt shear wave velocity (m/s)	
α_{p1}	Sediment compressional wave attenuation (dB/λ)	
α_{p2}	Basalt compressional wave attenuation (dB/λ)	
α_{s1}	Sediment shear wave attenuation (dB/λ)	
α_{s2}	Basalt shear wave attenuation (dB/λ)	
ρ_1	Density of sediment (kg/m^3)	
ρ_2	Density of basalt (kg/m^3)	
H	Sediment thickness (m)	
D	Number of variables/dimensions	All algorithms
ϵ	Objective value / cost	All algorithms
ϵ_i	Objective value / cost of solution number i	All algorithms
ϵ_{best}	Best objective value / cost in population	All algorithms
ϵ_{worst}	Worst objective value / cost in population	All algorithms
f	Objective function	All algorithms
$\mathbf{x}$	Input-vector / solution	All algorithms
$\mathbf{x}_i$	Input-vector / solution number i	All algorithms
$\mathbf{x}_{i,j}$	j'th variable of input-vector / solution number i	All algorithms
N_{pop}	Number of members in the population	GA,DE & ABC

N_{gen}	Number of generations run before termination	GA
N_{keep}	Number of population members kept each generation	GA
N_{mut}	Number of chromosomes mutated in each generation	GA
X_{rate}	Rate of the population that survives each generation	GA
μ	Mutation rate,	GA
CR	Crossover constant	DE
F_{weight}	Amplification of difference-vector	DE
$\mathbf{d}_i$	Difference-vector number i	DE
$\mathbf{m}_i$	Mutant-vector number i	DE
$\mathbf{r}_i$	Random-vector number i	DE
$\mathbf{tr}_i$	Trial-vector number i	DE
$limit$	Number of allowed searches without improvement	ABC
$maxCycle$	Maximum number of iterations	ABC
MR	Modification rate	ABC
$\mathbf{p}$	Probability-vector given by individual solution fitness	ABC
$\mathbf{r}$	Random-vector	ABC
SF	Perturbation magnitude	ABC
θ	Random value in the range $[-1, 1]$	ABC
FW	Fret width	HS
HM	Harmony memory	HS
HMS	Harmony memory size	HS
$HMCR$	Harmony memory considering rate	HS
$MaxImp$	Maximum allowed improvisation	HS
PAR	Pitch adjusting rate	HS
$\mathbf{x}_{new}$	New improvised harmony	HS
c_s	Scaling constant	DS
e_i	Unit-vector number	DS
P_0	Starting point	DS
P_i	Point number i	DS
P_n	Simplex point with lowest ϵ	DS
$\overline{P}$	Centroid of simplex	DS
P^*	Point found through reflection or contraction	DS

P^{**}	Point found through expansion	DS
α	Reflection coefficient	DS
β	Contraction coefficient	DS
γ	Expansion coefficient	DS
$L_{incfact}$	Increment factor of N_{local} for each hybrid iteration	Hybrids
N_{local}	Number of function calls performed by DS first hybrid iteration	Hybrids
N_{global}	Number of function calls performed by global optimizer each hybrid iteration	Hybrids

CONTENTS

Problem Description i

Abstract iii

Sammendrag v

Preface vii

Abbreviations ix

Symbols xi

1 Introduction 1
- 1.1 Global Optimization . . . 1
- 1.2 Global Optimization for Underwater Acoustic Applications . . . 2
- 1.3 Report Structure . . . 2

2 Global Optimization 5
- 2.1 Optimization . . . 5
  - 2.1.1 Optimization Categories . . . 6
  - 2.1.2 Notation . . . 7
- 2.2 Genetic Algorithm . . . 7
  - 2.2.1 GA Control Parameters . . . 8
  - 2.2.2 Genetic Algorithm-Step by Step . . . 9
    - 2.2.2.1 Selecting Control Parameters, Defining Variables and Objective Function . . . 11
    - 2.2.2.2 Create an Initial Population . . . 11
    - 2.2.2.3 Natural Selection . . . 11
    - 2.2.2.4 Mating . . . 12
    - 2.2.2.5 Mutation . . . 13
    - 2.2.2.6 Convergence Check . . . 13
- 2.3 Differential Evolution . . . 14

2.3.1 DE Control Parameters 14
2.3.2 DE-Step by Step 15
2.3.3 Selecting Control Parameters, Defining Variables and Objective Function 17
2.3.4 Choose a Random Initial Population 17
2.3.5 Mutation, Crossover and Selection 18
2.3.6 Convergence Check 19
2.4 Artificial Bee Colony 19
2.4.1 ABC Control Parameters 21
2.4.2 ABC-Step by Step 22
2.4.2.1 Selecting Control Parameters, Defining Variables and Objective Function 22
2.4.3 Choosing Initial Food Sources 24
2.4.4 Employed Bee Phase 24
2.4.5 Create Probability Distribution According to Population Ranking 26
2.4.6 Onlooking Bee Phase 27
2.4.7 Scout Bee Phase 27
2.4.8 Convergence Check 27
2.5 Harmony Search 27
2.5.1 HS Control Parameters 28
2.5.2 HS-Step by Step 29
2.5.2.1 Selecting Control Parameters, Defining Variables and Objective Function 31
2.5.3 Choose Initial Harmonies 31
2.5.4 Improvise a New Harmony 32
2.5.5 Check for Convergence 33
2.6 Downhill Simplex 33

3 Estimation of Geoacoustic Parameters **37**
3.1 Geoacoustic Parameters 37
3.2 Formulating the Inversion as an Optimization Problem 38

4 Implementation **41**
4.1 Implementing Global Optimization Algorithms 41
4.2 Creating Hybrid Algorithms 41
4.3 Finding the Control Parameters of the Global Optimizers 45
4.3.1 GA 46
4.3.2 DE 48
4.3.3 ABC 50
4.3.4 HS 52
4.4 Finding the Control Parameters of the Hybrids 54

5 Results and Discussion **61**
5.1 Variable Sensitivities 61

5.2 Testing Algorithm Speed 62
5.2.1 Rosenbrock Function 63
5.2.1.1 Setup 63
5.2.1.2 Results 63
5.2.2 Estimation of Geoacoustic Parameters 64
5.2.2.1 Setup 64
5.2.2.2 Results 65
5.3 Testing Algorithm Accuracy 65
5.3.0.3 Setup 65
5.3.0.4 Results 66
5.4 Discussion 68
5.4.1 Improvement of Including Adaptive Simplex 68
5.4.2 ASHS versus ASDE 68

6 Conclusion **75**

Bibliography **77**

CHAPTER 1

INTRODUCTION

1.1 Global Optimization

The search for an optimal state is one of the most fundamental principles in our world [24]. Optimization is used both in trivial daily situations and in advanced science and technology, business and economics. Global optimization makes it possible to find an optimal, or close to optimal, solution in a solution space with multiple local extrema.

Bio inspired algorithms are a type of algorithms which use processes inspired by biological evolution or swarm behavior to solve optimization problems. These processes are successful in optimizing natural phenomena [9]. Bio inspired algorithms include popular algorithms such as genetic algorithm [10], simulated annealing [16], differential evolution [21], ant colony optimization [5] and particle swarm optimization [20]. In this thesis some of these well-known algorithms will be compared with more recent global optimizers. The comparison will be done by testing the algorithms on a benchmark problem and on a problem of geoacoustic inversion.

1.2 Global Optimization for Underwater Acoustic Applications

Geoacoustic inversion is the process of estimating seabed geoacoustic properties from measured acoustic data. This is a topic that has received a lot of attention in recent research. Due to the huge number of possible parameter combinations, finding the parameters resulting in the recorded data is a difficult task. To solve this problem global optimization algorithms have been applied. Some of the most successful hybrids used in geoacoustic inversion are combinations of global bio inspired algorithms and Nealder Mead's downhill simplex local optimizer [11][17][6]. The use of local optimizers results in fast convergence, whereas the use of global optimizers keeps the hybrid algorithm from getting stuck in a local minimum. Examples of successful hybrids used in geoacoustic inversion are adaptive simplex simulated annealing (ASSA)[6] and adaptive simplex genetic algorithm (ASGA)[17]. Recently, adaptive simplex differential evolution (ASDE) [11] was implemented with even better results than ASGA and ASSA. ASDE is to date the fastest and most accurate algorithm used in geoacoustic inversion.

In recent years, powerful metaheuristic optimizers like harmony search (HS)[7], and artificial bee colony (ABC)[13], have been invented. In this thesis, HS and ABC, as well as their hybrid versions, adaptive simplex harmony search (ASHS) and adaptive simplex artificial bee colony (ASABC), will be adapted to the problem of estimating geoacoustic parameters of upper oceanic crust from synthetic ocean bottom reflective loss data. The first task of this thesis is to see whether the implementation of DS will increase the performance of the global optimizers. Then, their performance will be compared with ASDE which will be adopted and run on the same problem. The main goal of this project is to study if hybrid versions of these more recent algorithms can compare with the performance of ASDE. The algorithms will be compared both in speed, i.e. computational cost, and accuracy.

1.3 Report Structure

The next chapter will give a thorough presentation of the multiple optimization algorithms used in this thesis. Chapter 3 contains a brief explanation of geoacoustic parameter estimation, whereas Chapter 4 covers the implementation of

the algorithms and control parameter tuning. Chapter 5 includes test results and discussions of the matters introduced in this chapter. The conclusion of the thesis is placed in Chapter 6.

CHAPTER 2
GLOBAL OPTIMIZATION

2.1 Optimization

"Optimization is the process of adjusting the inputs to or characteristics of a device, mathematical process, or experiment to find the minimum or maximum output or result." [9] The goal of optimization is to find an optimal solution within the variable's bounds, i.e. inside the solution space. [24] Optimization is used in many fields such as economics, finance, acoustics, medicine and data analysis. Still, optimization can include basic operations such as driving the shortest route to work, or filling as much coffee as possible into a cup without spilling. All optimization problems have inputs, an objective function and an output. The input is a vector of length equal to the number of variables. Each element of the input-vector represents a variable value. The input-vector is in other words a solution found in the solution space. The objective function, also called fitness function or cost function, is the function that measures the given set of inputs. The output is the cost or fitness of the given inputs, and is often called objective value. A basic flowchart of optimization is illustrated in Figure 2.1.

Depending on the problem, the goal of optimization is either to maximize or minimize the output of an object function. To illustrate this, let's have a look on the coffee cup and route to work examples mentioned in the previous section. When choosing a route to work we want to minimize the distance. Whereas in the coffee cup example we want to maximize the amount of coffee filled in the cup.

Any maximization problem can however be solved as a minimization problem by setting a minus sign before the output. The coffee cup example transforms from maximizing to minimizing by shifting the problem to minimizing the volume in the cup not filled with coffee.

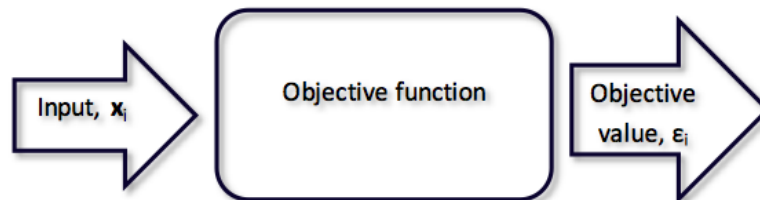

Figure 2.1: Flowchart of an optimization process where input is varied to achieve an optimal output.

2.1.1 Optimization Categories

There are many approaches to solving optimization problems. In analytical optimization calculus is used to solve the problems [9]. Metaheuristics are iterative methods that uses its own solutions and a set of rules to decide further actions [23]. Metaheuristic optimization can search very large solution spaces, but does not guarantee to find the best solution. Metaheuristics are preferred over analytical optimization if the solution space is very big, and the improvement of a solution is more important than finding the absolute best solution. Bio inspired algorithms are a type of metaheuristics that uses processes inspired by biological evolution or swarm behavior to solve optimization problems. These processes are successful in optimizing natural phenomena. For most metaheuristics, the biggest challange is to combine the right amounts of exploration and exploitation. Exploration is a quality that is necessary in order to search the entire solution space without getting stuck in a local optimum. Exploitation is the abiliy to use the previously tested variable combinations in order to quickly converge towards an optimum. Too much exploitation will lead to convergence towards a local optimum, while

too much exploration will lead to slow convergence, and thus a high computational cost. Balancing exploration and exploitation is the key to high performing metaheuristics [26].

2.1.2 Notation

Throughout this thesis the input-vector will be denoted by $\mathbf{x}$. In a collection of input-vectors, $\mathbf{x}_i$ represents input-vector number i and $\mathbf{x}_{i,j}$ represents the j'th variable of the i'th input-vector. The dimension of the problems are denoted by D. D is thus the length of the input-vector. When used as a subscript, $rand$ denotes a randomly picked integer in the range $[1, D]$. This means that $\mathbf{x}_{rand}$ is a random solution chosen from the population. In all other uses, $rand$ is a random number in the range $[0, 1]$ drawn with an uniform probability distribution. The objective function is denoted by f and the objective value ϵ. This gives $\epsilon = f(\mathbf{x})$. Further will ϵ_i represent the objective value of $\mathbf{x}_i$. I.e. $\epsilon_i = f(\mathbf{x}_i)$. In most cases the algorithms are terminated if ϵ reaches a tolerated objective value. This value is denoted by ϵ_{tol}.

The following sections will contain some basic theory for the genetic algorithm (GA), differential evolution (DE), artificial bee colony (ABC), harmony search (HS) and downhill simplex (DS).

2.2 Genetic Algorithm

The genetic algorithm (GA) is an optimization technique that is based on nature's own ability to evolve and move towards better solutions [24]. Imagine the population of a species restricted to a finite number of members due to food constraints. The individuals who possess the qualities best suited for survival, in other words the most fit, will survive and mate with each other. The least fit members of the population will eventually die. The offspring of the fit individuals will inherit qualities from both of their parents. This results in new combinations of genes. Some of these new combinations are even better than the gene combination of their parents. As the new generation grows up, the most fit individuals survive and mate with each other. Thus the population has a chance of getting more fit members for each generation. Every once in a while a mutation happens. As a

result, some of the population members genes do not resemble the genes of their parents. In some cases these mutations result in a less suitable combination of genes. In other cases however, the mutation adds a new property which proves to be beneficial to the individuals who possess it. If this is the case, the mutated gene will be passed on to future generations.

GA tries to imitate the process described above. Each individual of the population resembles a solution and is called a chromosome. A set of randomly picked solutions (chromosomes) are rated by using the chromosomes as inputs to the objective function of the problem. The chromosomes consist of specific values for each of the problems variables. These values are called genes. The least fit chromosomes are deleted and the rest are paired for mating. As a result of the mating a new set of chromosomes are produced. As in nature, these chromosomes inherit qualities from both of their parents. After mating mutation is implemented by switching some of the chromosomes genes with random values from the solution space. This prevents the GA from converging towards a local minimum. In this way the algorithm will hopefully converge towards the global optimum, instead of a local.

2.2.1 GA Control Parameters

GA can be adjusted by a set of control parameters. Wisely chosen control parameters will increase the probability of converging towards a global optimum.

The population size, N_{pop}, is the number of chromosomes in the population. A large population size will help to search all parts of the solution space. The number of times GA operations such as ranking, mating and mutating is proportional to the population size. A large population size will thus reduce the convergence speed.

The mutation rate, μ, is the rate of cells that will be mutated. If $\mu = 0.2$, 20% of the populations genes will be mutated. Mutations are implemented to avoid converging against local maxima. By replacing random genes with values from other parts of the solution space, global optimization can be obtained.

The selection rate, X_{rate} is the rate of the population that survives each generation. This means that $1 - X_{rate}$ chromosomes will be terminated each generation.

2.2.2 Genetic Algorithm-Step by Step

The flow of GA is illustrated in Figure 2.2. Each of these stages will be explained through the following example.

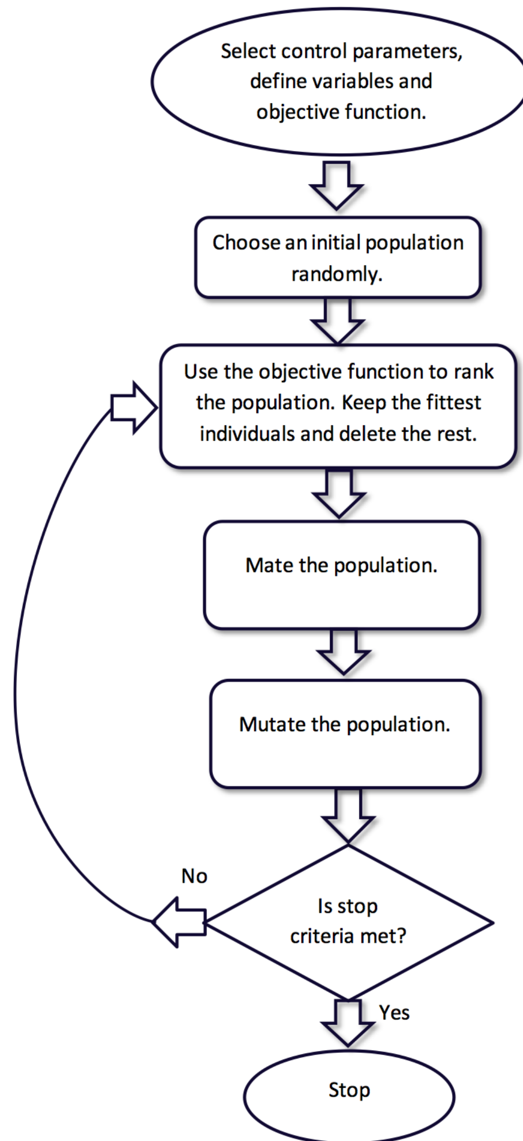

FIGURE 2.2: Flowchart of GA.

Let's say we want to find the maximum value of the function

$$f(x, y) = y\sin(4x) + 1.1x\sin(2y) \qquad \forall x, y \in [0, 10] \tag{2.1}$$

This problem is illustrated in Figure 2.3.

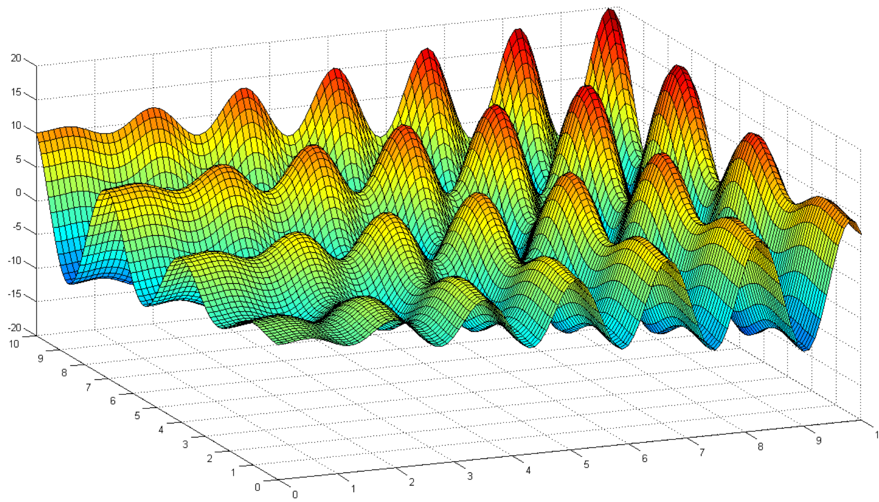

FIGURE 2.3: 3-dimensional landscape. We want to find the highest point by using GA.

As stated earlier, this is the same as finding the minimum of the function

$$f(x, y) = -(y \sin(4x) + 1.1x \sin(2y)) \qquad \forall x, y \in [0, 10] \tag{2.2}$$

This particular problem only has two variables and can be solved by less complex methods than the GA. Still, it is a good example for explaining the basics.

2.2.2.1 Selecting Control Parameters, Defining Variables and Objective Function

The first thing we need to do is to select control parameters, define the variables and the objective function. The objective function is already given by equation (2.2). In this example the variables are x and y. They are both bound by zero below and ten above. By varying x and y inside of these bounds, equation (2.2) will give us a z value which is the objective value, ϵ of the function. The goal of this optimization is to find the lowest z value possible . At this point we also need to set the control parameters. In this example we will use
$N_{pop} = 8$
$X_{rate} = 0.5$
$\mu = 0.2$

2.2.2.2 Create an Initial Population

Now it's time to create the initial population. In our example the initial population consists of 8 pairs of x and y values between zero and ten. These are chosen randomly. Since both variables have the same bounds, this can be done with $rand(N_{pop}, 2) = 10 * rand(8, 2)$. The initial population is showed in Table 2.1.

x	y	ϵ
4.22	6.79	-2.30
9.16	7.58	-1.35
7.92	7.43	8.52
9.59	3.92	13.02
6.56	6.55	9.56
0.36	1.71	1.59
8.49	7.06	13.28
9.34	0.32	6.00

Table 2.1: Randomly chosen initial population with corresponding objective value.

2.2.2.3 Natural Selection

In this part of GA, the population is ranked, and the most fit chromosomes are selected to survive. N_{keep} is the number of chromosomes that survives this stage.

x	y	ϵ
4.22	6.79	-2.30
9.16	7.58	-1.35
0.36	1.71	1.59
9.34	0.32	6.00

Table 2.2: Population after natural selection.

$N_{keep} = N_{pop} * X_{rate}$. The least fit chromosomes are deleted. An easy way to do this is to rank the population and keep the N_{keep} most fit chromosomes. This is the approach used throughout this thesis. The result of the natural selection is showed in Table 2.2.

2.2.2.4 Mating

Now that our population only consists of the N_{keep} most fit chromosomes, it's time to pair them up for mating. From each mating, two children will be produced. Thus we need two pairs of parents. There are many ways to achieve this, but in this example we will choose an easy approach. Until the needed number of pairs is reached, the first chromosome will mate the second, the third will mate the fourth and so on. In our example only two matings will be performed.

The mating itself can be done in numerous ways. In this thesis single-point crossover will be used. The first child, $\mathbf{x}_{child1}$ inherits the x value from $\mathbf{x}_1$ and the y value from $\mathbf{x}_2$. The second child, $\mathbf{x}_{child2}$ will inherit the y value from $\mathbf{x}_1$ and x value from $\mathbf{x}_2$. This is illustrated below.

$$\begin{aligned}
\mathbf{x}_1 &= [4.22, 6.79] \\
\mathbf{x}_2 &= [9.16, 7.58] \\
\mathbf{x}_{child1} &= [4.22, 7.58] \\
\mathbf{x}_{child2} &= [9.16, 6.79]
\end{aligned}$$

This mating process is repeated for the third and fourth most fit chromosome, i.e. $\mathbf{x}_3$ and $\mathbf{x}_4$ in Table 2.2. The children will take the places five to eight in the population.

2.2.2.5 Mutation

The GA described so far works great for finding local maximum. It does however not perform well in finding global maxima. If all the chromosomes in the initial population are close to a local maximum, chromosomes ranked as most fit will be the ones closest to this maximum. Thus the algorithm will converge against this point.

To solve this problem, mutation is included. Mutation is the process of giving randomly picked genes a randomly picked value within that specific variable's bounds. This helps GA test completely new combinations of genes. In our example the mutation rate, μ, is set to 0.2. This means that 20% of the genes in the population will be mutated. Often the most fit solution is spared from mutation. In this way the best solution of the next generation can not be less fit than the best chromosome in the current generation. This is referred to as elitism. In our example the number of mutations, N_{mut}, is given by equation (2.3)

$$N_{mut} = ceil((N_{pop} - 1) * D * \mu) = ceil(7 * 2 * 0.2) = 3 \tag{2.3}$$

where D describes the number of variables. Thus we need to pick N_{mut} genes randomly, and give them a random value within their bounds. This is illustrated by equation (2.4)

$$\mathbf{x}_{rand,rand} = ub \times rand(ub - lb) \tag{2.4}$$

where $\mathbf{x}_{rand,rand}$ is a randomly picked variable in a randomly picked population member. ub represents the upper bound and lb represents the lower bound of the randomly picked variable.

In our example the number of mutations is three. Thus we need to perform equation (2.4) three times.

2.2.2.6 Convergence Check

The steps from natural selection to mating are done until either the objective value of the most fit chromosome is less than a threshold, or when the number

of generations reaches a predefined maximum. GA can also be terminated if the progress made over a given number of generations is to small.

In our example the minimum $\epsilon = f(0.9039, 0.8668) = -18.5547$ was found after 28 generations. The answer to this optimization problem is thus

$$x = 0.90039 \quad , \quad y = 0.8668 \tag{2.5}$$

2.3 Differential Evolution

Differential evolution (DE) is another evolution based global optimizer. Like GA, DE makes use of operations like selection, crossover, and mutation on a population of solutions. New solutions are generated by adding the weighted difference between two solutions, randomly picked from the population, to a third solution in the population [21]. This mutant-vector is thus mixed with a fourth vector, the target-vector, through crossover. The resulting solution is called the trial-vector. If the trial-vector yields a lower objective value than the target-vector, the trial-vector takes the target-vector' s place. For each generation every population member serve as target-vector once. This means that N_{pop} competitions takes place each generation.

2.3.1 DE Control Parameters

The size of the population used in differential evolution is like GA controlled by the parameter N_{pop}.

As mentioned above, the trial-vector is created as a mix of the mutant-vector and another vector picked randomly from the population. This process is controlled by the crossover constant, $CR \in [0, 1]$. For each variable in the vector, the crossover constant determines the probability of choosing the given variable from the mutant-vector. This means that if CR is given a large value, the target-vector will most likely consist of mostly values from the mutant-vector. The opposite is true if CR is given a small value. The effect of a high-valued CR is similar to the effect of a high-valued mutation rate in the genetic algorithm. Giving CR a large

value will help the algorithm to cover a larger area of the search space. This will however be on the expense of convergence speed.

F_{weight} controls the amplification of the difference between the two randomly picked vectors chosen from the population. If $F_{weight} = 0$, the difference between the two randomly picked vectors will be neglected, and the trial-vector will simply consist of the third randomly chosen vector. This will results in no new solutions in the population as no perturbations would ever be made to the initial population. If $F_{weight} = 2$ however, the perturbation made to the members of the population will be greater. Thus, F_{weight} controls the algorithms mutation, but unlike CR it is the magnitude of the mutations that is controlled, and not the frequency.

2.3.2 DE-Step by Step

The flow of differential evolution is given in Figure 2.4. As in the GA-section, these steps will be explained through the example where the goal is to find the minimum of equation (2.2). The variables are still x and y and their bounds are given by equation (2.2).

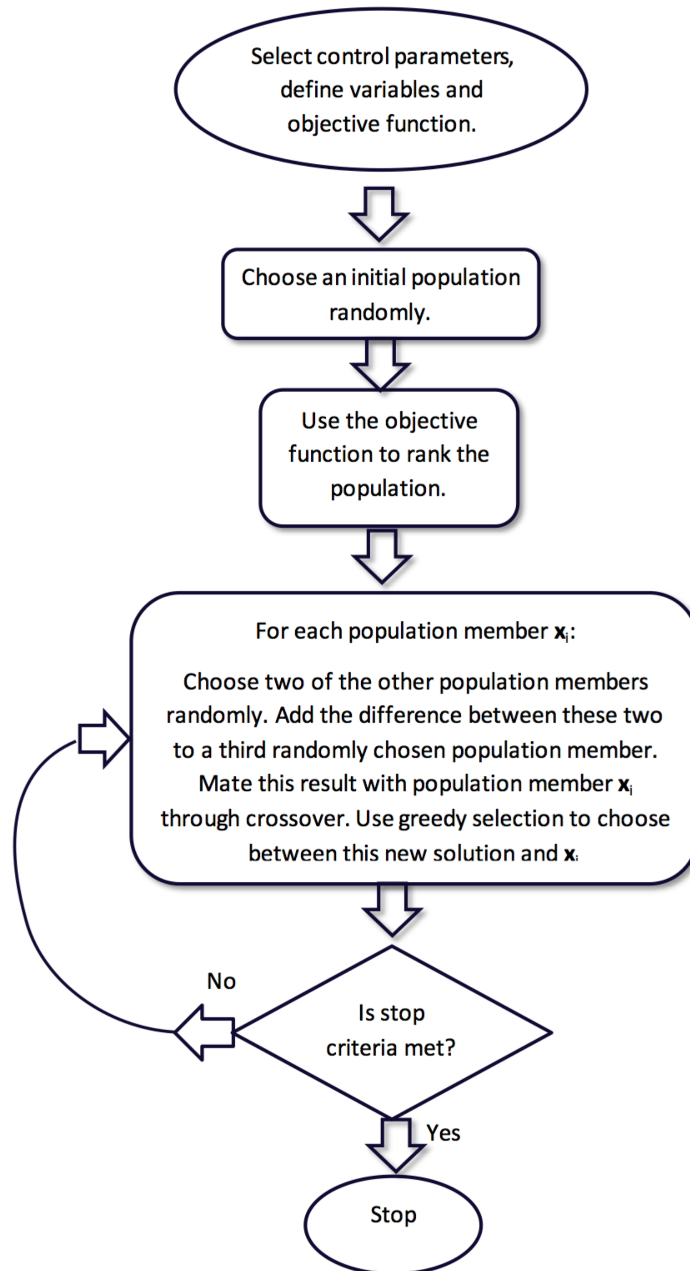

Figure 2.4: Flowchart of DE.

2.3.3 Selecting Control Parameters, Defining Variables and Objective Function

This step is almost identical to the first step of GA. The only difference is in fact the control parameters. In this example we will use the following control parameters:

$N_{pop} = 8$
$CR = 0.8$
$F_{weight} = 0.5$

2.3.4 Choose a Random Initial Population

The creation of a random initial population, happens in exactly the same way as for the genetic algorithm. N_{pop} population members are given D variables, which are randomly picked with an uniform distribution over the range of each variable. The population is then ranked by letting each population member be the input to the objective function. Table 2.3 shows the initial population of our example sorted by objective value.

x	**y**	ϵ
4.98	6.99	-11.60
6.55	7.51	-11.14
7.09	9.60	-1.68
7.55	3.40	-0.95
1.63	2.55	1.09
6.80	2.24	5.29
1.19	5.06	5.89
2.76	5.85	8.15

Table 2.3: Randomly chosen initial population with corresponding objective value. The population is ranked according to objective value.

2.3.5 Mutation, Crossover and Selection

Let's further call the vector containing the solution placed in the first row of the population $\mathbf{x}_1$, the second $\mathbf{x}_2$ and so on. For each population member, $\mathbf{x}_i$, a trial-vector, $\mathbf{tr}_i$, will be created. This is done by randomly selecting two population members, $\mathbf{x}_a$ and $\mathbf{x}_b$, where $a, b \neq i$. From these vectors a difference vector is created by subtracting one of the vectors from the other: $\mathbf{d}_i = \mathbf{x}_a - \mathbf{x}_b$. This difference vector is thus multiplied with the control parameter F_{weight} and added to a third randomly picked population member, $\mathbf{x}_c$ where $c \neq i, a, b$. The results of this process is the mutant-vector, $\mathbf{m}_i$. Mathematically we have $\mathbf{m}_i = \mathbf{x}_i + F_{weight}\mathbf{d}_i$. The mutant-vector is thus mixed with the target-vector, $\mathbf{x}_i$, through crossover. In the version of differential evolution used in this thesis, crossover is done by the following equation (2.6).

$$\mathbf{tr}_{i,k} = \begin{cases} \mathbf{m}_{i,k} & \text{if } \mathbf{r}_{i,k} \leq CR \\ \mathbf{x}_{i,k} & \text{if } \mathbf{r}_{i,k} > CR \end{cases} \qquad k = 1, 2, ..., D \tag{2.6}$$

where $\mathbf{r}_i$ is a vector of length D containing random values between zero and one. $\mathbf{r}_{i,k}$ represents the k'th element of $\mathbf{r}_i$. Equation (2.6) tells us that the k'th value of the trial-vector will contain the k'th value of the mutant-vector if the k'th value of the random-vector is smaller than or equal to CR, and the k'th value of the target-vector, $\mathbf{x}_i$, if the k'th value of the random-vector is greater than CR. After mutation and crossover, DE calculates the objective value of the trial-vector, and chooses between $\mathbf{tr}_i$ and $\mathbf{x}_i$ by using greedy selection.

Let's perform these steps on the first population member of our example, $\mathbf{x}_1$. After randomly choosing values for a, b and c, we have the following set of vectors:

$$\mathbf{x}_1 = [4.98, 6.99]$$
$$\mathbf{x}_a = \mathbf{x}_4 = [7.55, 3.40]$$
$$\mathbf{x}_b = \mathbf{x}_7 = [1.19, 5.06]$$
$$\mathbf{x}_c = \mathbf{x}_2 = [6.55, 7.51]$$

We start with calculating the difference-vector:

$$\mathbf{d}_1 = \mathbf{x}_a - \mathbf{x}_b = [7.55, 3.40] - [1.19, 5.06] = [6.36, -1.66] \tag{2.7}$$

Further we achieve the mutation-vector:

$$\mathbf{m}_1 = \mathbf{x}_1 + F_{weight}\mathbf{d}_1 = [4.98, 6.99] + 0.5[6.36, -1.66] = [8.16, 6.16] \tag{2.8}$$

The final step is the crossover, where $\mathbf{x}_1$ and $\mathbf{m}_1$ will be mixed. In order to do that we need to generate the random-vector, by choosing one random value between zero and one for each variable of the problem. We get $\mathbf{r}_i = [0.43, 0.96]$. Crossover is then performed according to equation (2.6). Remember that we have already chosen $CR = 0.8$. We see that $\mathbf{r}_{1,1} = 0.43 < CR$. The first element of the trial-vector thus get the value $\mathbf{tr}_{1,1} = 8.16$. Since $0.96 > CR$, the second element of the trial-vector gets the value $\mathbf{tr}_{1,2} = 6.99$. In order to choose between $\mathbf{x}_1$ and the trial-vector, we need to calculate the objective value of the trial-vector. We already know that $\epsilon_1 = f(\mathbf{x}_1) = -11.60$. By letting $\mathbf{tr}_1$ be the input to equation (2.2), we find that $\epsilon_{tr1} = f(\mathbf{tr}_1) = -15.44$. Since $\epsilon_{tr1} < \epsilon_1$, the trial vector is selected by greedy selection, and thus replaces the target vector in the population.

These steps of mutation, crossover and selection is repeated for all the members of the population. The steps require one target-vector, two vectors to create the difference-vector and one a fourth vector that the difference-vector will be added to. Since all of these vectors have to be unique, differential evolution requires a minimum of four members in the population.

2.3.6 Convergence Check

When mutation, crossover and selection has been applied to the entire population, one generation is complete. The algorithm will then check if any stop criteria is met. If not, mutation, crossover and selection is applied to the new generation.

In our example the minimum $\epsilon = f(0.9039, 0.8668) = -18.5547$ was found after 23 generations.

2.4 Artificial Bee Colony

Artificial Bee Colony is inspired by the way bee colonies search for the best food sources in an area [13]. The value of a food source is dependent on multiple factors

such as proximity to the hive, the amount of nectar and the ease of extracting the nectar. The bee colony solve this global optimization problem by dividing the bees into three groups: employed bees, unemployed bees and scouts.

The employed bees are employed at a particular food source. When returning to the hive the bees carry both the extracted nectar and information about the position and quality of their food source. They will share this information with other bees with a probability dependent on the value of the food source.

The unemployed bees, also called onlooker bees, are bees that don't have a food source to exploit. The unemployed bees uses information given by the employed bees to choose a food source of high quality to exploit.

Scouts search randomly after food sources. When a food source is exhausted, the employed bees at the food source becomes scouts.

The Artificial Bee Colony algorithm is inspired by this process. The position of a food source represents a solution/input-vector and the quality of the food source represents the objective value of the solution. A food source can only be explored once, thus each food source is only exploited by one employed bee. This means that the number of employed bees equals to the number of food sources i.e. the number of solutions in the population.

The food sources currently being exploited by employed bees form a population of food sources. We use $\mathbf{x}_i$ to represent food source number i. For each iteration the employed bees investigate the nectar amount of a food source close to the one they currently exploit. For every variable of each food source, ABC chooses randomly if the variable should be altered or not. Each variable that are chosen to be altered use equation (2.9) to find a new variable value.

$$\mathbf{x}_{i,j(new)} = \mathbf{x}_{i,j(old)} + \phi * (\mathbf{x}_{i,j(old)} - \mathbf{x}_{rand,j}) \tag{2.9}$$

In equation (2.9) $\mathbf{x}_{rand,j}$ is variable number j of a randomly chosen food source in the population. ϕ is a randomly chosen value in the range $[-1, 1]$. This means that the new variable value is a combination of variable values within the population of food sources.

2.4.1 ABC Control Parameters

ABC can like many other metaheuristics be controlled by its population size, N_{pop}. The population size includes both employed and unemployed bees. Half the population consists of employed bees, and the other half consists of unemployed bees. Increasing the population size will make the employed phase and the unemployed phase longer. This means that more solutions are tested before information between the bees are shared.

If a solution has not been improved after a given number of mutations, the food source is abandoned. This number of mutations is controlled by the parameter *limit*. When the area around a good solution has been tested *limit* times, the area is abandoned. If *limit* is exceeded, the employed bee exploiting the area becomes a scout, searching randomly after a food source of higher quality. If *limit* is set to low, there is a higher risk of missing a global optimum since each area in the solution space possibly containing a global optimum is not searched thoroughly enough. Too high *limit* will however be on the expense of the convergence speed. Based on [13] limit will be given a value equal to the population size multiplied with the dimension of the problem. $limit = N_{pop} \times D$

$maxCycle$ is a value describing the maximum allowed iterations. One iteration includes the employed bee phase, unemployed bee phase and scout bee phase. If $maxCycle$ is given a high value, ABC will be more expensive in terms of computational power, but will have a higher probability of finding the global optimum. If the optimization process is stuck in a local optimum, increasing $maxCycle$ will not enhance performance. Increasing $maxCycle$ can however not affect the problem output negatively.

Based on recent research on ABC solving real-valued problems [14], two more parameters have been included. They are called modification rate, MR, and perturbation magnitude, SF. For every variable in every food source a random value in the range $[0, 1]$ is generated. A variable is further modified if it's corresponding random value is smaller than MR. This means that giving MR a high value will increase the probability of modifying each variable. The value given to the modification rate has to hold the condition $0 < MR < 1$. Whereas the modification rate controls the frequency of variable modification, SF controls the magnitude of deviation. When the randomly generated value is lower than MR, the given

variable value is multiplied with a randomly generated value between $-SF$ and SF. The modified variable are given a new value from equation (2.10)

$$\mathbf{x}_{i,j(new)} = \mathbf{x}_{i,j} + (-SF + 2SF \times rand) \times (\mathbf{x}_{i,j} - \mathbf{x}_{rand,j}) \tag{2.10}$$

In the modified version of ABC used in this thesis, equation (2.9) is replaced by equation (2.10). The magnitude of SF will affect the magnitude of deviation from the value of the current food sources given variable. As long as the number of improvement trials of a current solution, i.e. the number of checked food sources in the neighboring area of a successful food source, is below *limit* trials, ABC will inspect a food source in the area close to the successful food source. SF decides how large this neighboring area is. Since modification rate and perturbation magnitude controls the deviation from successful solutions, they control the trade-off between exploration and exploitation.

2.4.2 ABC-Step by Step

This section will provide a step by step walk-through of the ABC. The flow of the algorithm is displayed in Figure 2.5.

2.4.2.1 Selecting Control Parameters, Defining Variables and Objective Function

The initial step of the ABC is to select control parameters and defining the variables and objective function. In this step-by-step walk-through the maximization problem illustrated in Figure 2.3 will be solved. This means that our objective function is given by equation (2.2). Then a set of control parameters are chosen. In this example we will use

N_{pop}= 16
$limit = D \times n_{pop}$=$2 \times 16 = 32$
$maxCycle$=500
MR=0.3
SF=0.5

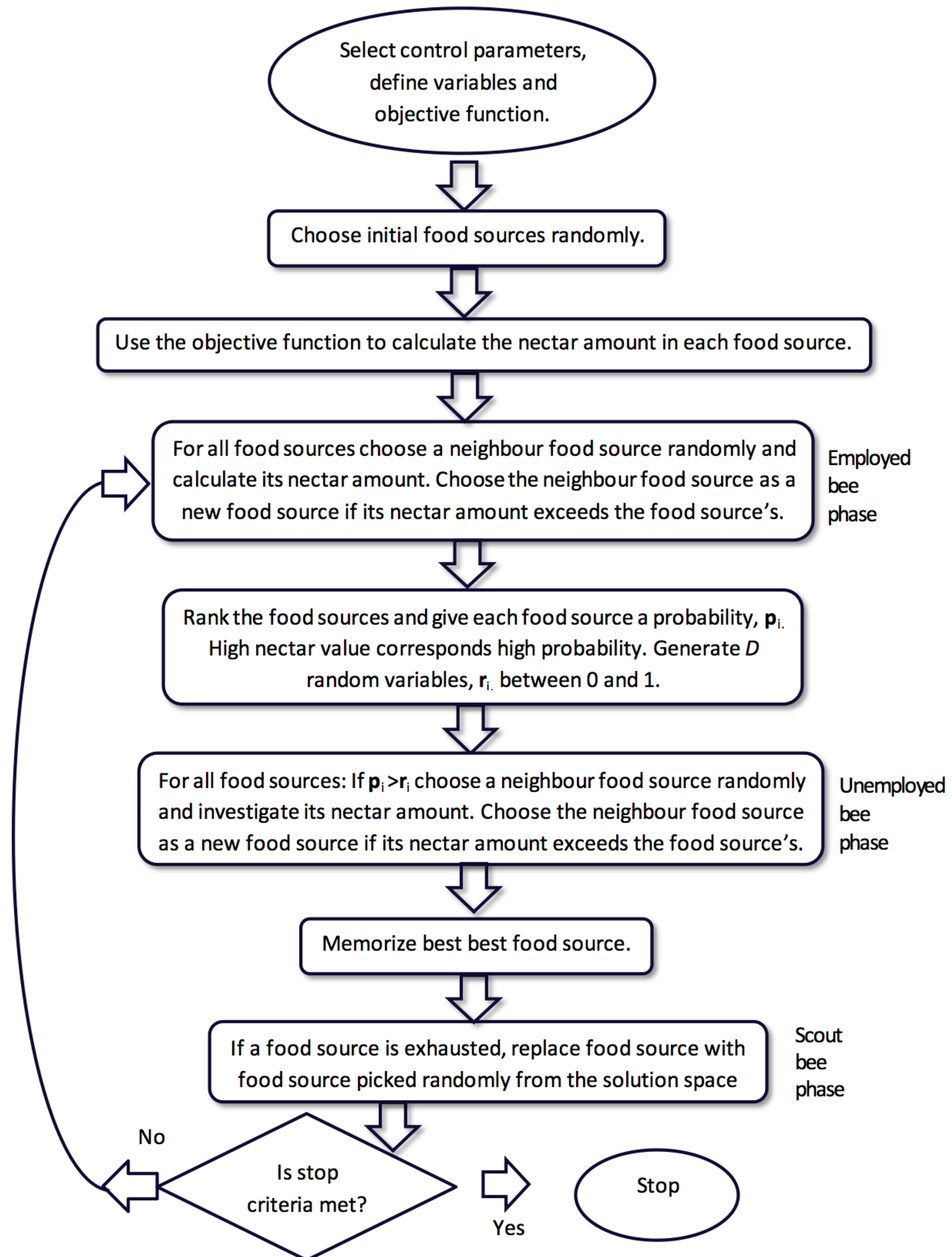

Figure 2.5: Flowchart of ABC.

2.4.3 Choosing Initial Food Sources

A set of initial food sources are randomly picked from the solution space. Since $N_{pop} = 16$, we have 8 food sources. Each food source consists of an x and an y value. The objective function is used to calculate the nectar amount of each food source. The nectar amount is negatively related to the objective value of a food source. The randomly chosen initial food sources and their objective values are shown in Figure 2.4.

x	**y**	ϵ
6.32	9.57	-3.55
0.98	4.85	3.64
2.78	8.00	8.82
5.47	1.42	-1.96
9.58	4.22	-11.19
9.65	9.16	-1.74
1.58	7.92	0.07
9.71	9.59	-12.22

TABLE 2.4: Randomly chosen initial food sources.

2.4.4 Employed Bee Phase

For each of the current food sources in the population, a near by food source is investigated for nectar amount. We will denote this near by food source with an asterisk. Then the near by food source of food source number i is denoted $\mathbf{x}_i^*$. Lets start with the food source in the first row of Table 2.4. For each of the two variables, a random value between 0 and 1 is generated. If a variable is to be modified, this random value has to be smaller than MR, in our case, 0.3. Say the two random values are: $MRrand_1 = 0.47$
$MRrand_2 = 0.24$
Since $MRrand_1$, which corresponds to the x-value, is above MR, the x-value will remain unchanged. $MRrand_2$, corresponding to the y-value, is however below MR. This means that the y-value of $\mathbf{x}_1$ should be modified by using equation (2.10). Before this value can be calculated, we have to randomly choose one of the food sources in the population. In this case $\mathbf{x}_4$ was selected, meaning that $\mathbf{x}_{rand,j}$ in equation (2.10) will be substituted by the y-value of the fourth food source in the population. Thus equation (2.10) becomes

$$\mathbf{x}^*_{1,2} = 9.57 + (-0.5 + 2 * 0.5 * rand) * (9.57 - 1.42) \tag{2.11}$$

Since the random value generated by the random generator *rand* can vary greatly, the result of equation (2.11) can have a wide range of values. In our case, the resulting variable-value became 5.78. The new food source is thus $\mathbf{x}^*_1 = [6.32, 5.78]$. Now it's time to check if the modified food source has higher nectar amount than its original. Before these food sources are compared, their fitness is calculated. The fitness is negatively proportional with the objective value and is given by equation (2.12). The process of calculating the fitness is always positive, and is thus easier to handle in the next steps of the algorithm.

$$fitness_i = \begin{cases} \frac{1}{1+f(\mathbf{x}_i)} & \text{if } f(\mathbf{x}_i) \leq 0 \\ 1 + abs(f(\mathbf{x}_i)) & \text{if } f(\mathbf{x}_i) \geq 0 \end{cases} \tag{2.12}$$

By replacing $\mathbf{x}_i$ in equation (2.12) with $\mathbf{x}^*_i$ we find the fitness of the near by food source, $fitness^*_i$. After inserting $\mathbf{x}_i$ and its near by food source, $\mathbf{x}^*_i$, into this equation we get.

$fitness_1 = 0.23$
$fitness^*_1 = 0.17$

Since $fitness_1 > fitness^*_1$ the nearby solution was not an improvement. $\mathbf{x}_1$ is thus kept, and $\mathbf{x}^*_1$ is discarded.

The above process is repeated for every food source in the population. The resulting population after the first employed bee phase, including cost and fitness is presented in Table 2.5 It can be observed that modifications has been made only to the x-value of the third and fourth food source, and the y-value of the seventh food source.

x	**y**	ϵ	**fitness**
6.32	9.57	-3.55	4.55
0.98	4.85	3.64	0.22
3.42	8.00	-6.03	7.03
3.65	1.42	-2.46	3.46
9.58	4.22	-11.19	12.19
9.65	9.16	-1.74	2.74
1.58	7.49	-1.31	2.31
9.71	9.59	-12.22	13.22

Table 2.5: Population of food sources after first employed bee phase.

2.4.5 Create Probability Distribution According to Population Ranking

The employed bees give the onlooking bees information about the quality of the food sources. Based on the quality of a food source, relative to the other food sources in the population, there is a certain probability that an unemployed bee will exploit the given food source. In ABC each food source is thus given a probability value based on its population ranking. This probability is further used in the unemployed bee phase. These probability values can be calculated in numerous ways. In this thesis they are calculated by equation (2.13)

$$prob_i = \frac{0.9 * fitness_i}{max(fitness)} + 0.1 \tag{2.13}$$

The probability-vector is in our example generated by using equation (2.13) on the fitness values in Table 2.5. Before we proceed to the unemployed bee phase, a vector of length N_{pop} containing random values between zero and one. The resulting two vectors are shown below.

$\mathbf{p} = [0.41, 0.12, 0.58, 0.34, 0.93, 0.29, 0.26, 1]$
$\mathbf{r} = [0.57, 0.94, 0.13, 0.42, 0.38, 0.84, 0.14, 0.68]$

2.4.6 Onlooking Bee Phase

In nature, the unemployed bees choose what areas to search based on the information given by the employed bees. In ABC this is done with the probability-vector, $\mathbf{p}$, and the random-vector, $\mathbf{r}$, created in the previous section. An unemployed bee will search the area close to food source number i if $\mathbf{p_i} > \mathbf{r_i}$. The unemployed bee will then investigate a near by food source, $\mathbf{x}_i^*$, in just the same way as the employed bees.

2.4.7 Scout Bee Phase

If a food source has been mutated over *limit* times without any progress, the food source is exhausted. The bee employed at the food source then becomes a scout. The scout investigates a random solution. This new solution is accepted even if the corresponding fitness is lower than the exhausted food source. For this reason the best food source and its corresponding objective value has to be memorized before the scout bee phase can start.

2.4.8 Convergence Check

If any stop criteria is reached, the algorithm is terminated. If not a new iteration is performed by going back to the employed bee phase. In our example the minimum $\epsilon = f(0.9039, 0.8668) = -18.5547$ was found after 25 generations.

2.5 Harmony Search

In music, a harmony is defined as simultaneously played musical notes. Harmony Search (HS) is based on improvising musicians search for good sounding harmonies [7]. Let's say three musicians are improvising. Each harmony then consists of three notes. The improvising musicians try to find a good sounding harmony, and thus maximizing the quality of the harmony. For each note each musician has two main choices. He can play a random note or play a note that has been a part of a previously successful harmony, which is recalled from his memory of harmonies. If he chooses a note from a previously successful harmony he has the choice of

playing it like he did last time, or slightly adjust the note randomly. The resulting harmonies is thus a combination of completely new notes, notes that has been successful in the past and small alterations of notes that has been successful in the past.

In HS the harmonies corresponds to solutions of the optimization problem. Each of the notes correspond to a variable. The quality of the harmony is calculated by the objective function. Successful harmonies are stored in *harmony memory*, HM. For each iteration a new solution is created as a combination of successful harmonies stored in HM and new variable values generated randomly. If the new harmony has lower objective value than the worst harmony in HM, the new harmony taktes the worst harmony's place.

2.5.1 HS Control Parameters

Like most metaheuristics, HS has a number of control parameters that controls the performance of the optimization.

Similar to the other three algorithms covered so far, HS has a control parameter for setting the size of the population. In HS the population is referred to as harmony memory. The size of the harmony memory is set by the control parameter harmony memory size, *HMS*. In other words: the best HMS harmonies found so far is placed in the harmony memory. The HMS parameter is very similar to the population size, N_{pop} in GA, DE and ABC. Creating a harmony based on a previous harmony can be thought of as exploring the solution space in the area close to a previous successful harmony. Thus will a small HMS lead to exploration of only a few areas at the time. On one hand this can lead to fast convergence. On the other hand we might get stuck in a local minimum.

The number of improvisations, i.e. iterations, made before termination is controlled by the parameter $MaxImp$. HS can be terminated before it has reached $MaxImp$ improvisations only if the accepted optimization error is reached. The objective function is called only once during an iteration. This separates HS from GA, DE and ABC where the number objective function calls during an iteration is correlated with the population size.

For each note in a new harmony there is a given chance that it will be based on a note from a previous harmony. This probability is set by the parameter harmony

memory considering rate, $HMCR$. If $HMCR$ is set to 0.9 there is a 90% chance that a new note will be based on a note from the harmony memory. A high value for $HMCR$ will result in good exploiting abilities, whereas a low value will result in good exploring skils.

When new harmonies are created and a note randomly has been chosen to be based on a note in the harmony memory, the new note can either be an exact copy of the note in HM or it could be a slight alteration of it. This is decided randomly, but is controlled by the parameter pitch adjusting rate, PAR. If PAR is set to 0.2 there is a 20% chance that each note picked from HM will be altered. This parameter resembles mutation in GA.

When a note is an alteration of a previous successful note, the amount of deviation from the previous note is controlled by the parameter fret width, FW. The new note is given a random value which maximum deviates by FW from the successful note. The new note is given as $\mathbf{x}_{new,i} = \mathbf{x}_{rand,i} + rand \times FW$, where i is the given note's index in the harmony.

2.5.2 HS-Step by Step

To understand the flow of HS this section will provide a step by step explanation of the algorithm. A flowchart of HS is given in Figure 2.6. As the other algorithms this will be done while finding the highest point in the 3-dimensional landscape given by Figure 2.3 and equation (2.2).

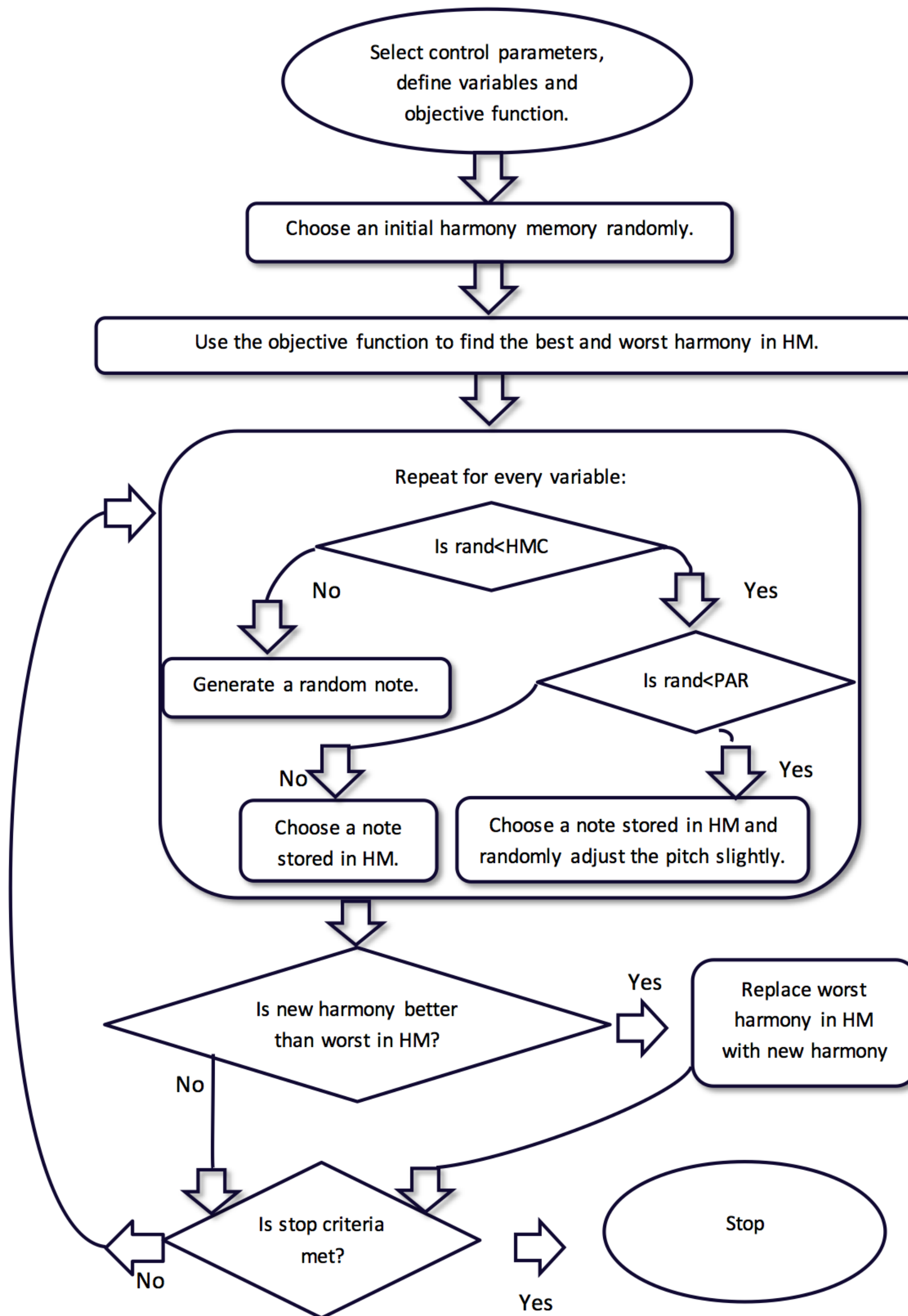

Figure 2.6: Flowchart of HS.

2.5.2.1 Selecting Control Parameters, Defining Variables and Objective Function

As GA and ABC the first thing we need to do is to select control parameters, define the variables and objective function. Both the objective function and variables will be the same as GA and ABC. The control parameters are however not the same. In this example we will use
$HMS = 8$
$MaxImp = 10000$
$HMCR = 0.8$
$PAR = 0.2$
$FW = (Ub - Lb)/100$

2.5.3 Choose Initial Harmonies

The first thing that needs to be done is to fill the harmony memory with randomly selected harmonies within the variable bounds. In this case each harmony consists of a pair of x and y values. Since $HMS = 8$ there will be a total of 8 randomly chosen harmonies. These harmonies are then evaluated by the objective function. The initial harmony memory is shown in Table 2.6.

x	**y**	ϵ
1.49	6.75	0.82
9.94	8.45	2.81
6.44	6.20	-2.47
2.91	7.37	3.19
0.54	8.83	-6.88
3.22	3.78	-4.59
7.41	9.14	13.32
2.28	1.55	-0.60

Table 2.6: Randomly chosen initial harmony memory.

In HS the solutions are not ranked from best to worst. In order to check if a stop criteria is met, we need to know the objective value of the best solution. As mentioned, a new improvised harmony will take the place of the worst harmony in the harmony memory if its objective value is lower than the worst harmony.

Thus we also need to know the objective value of the worst solution. The index of these harmonies are also necessary to know in order to know the variable values that make the harmony. In our example the best and worst solutions are
$\epsilon_{best} = -6.88$
$\epsilon_{worst} = 13.32$
These objective values represent the fifth and the seventh harmony respectively, i.e. $\mathbf{x}_5$ and $\mathbf{x}_7$.

2.5.4 Improvise a New Harmony

After the initialization it's time to create some new harmonies. The harmonies are created one by one. A new harmony is not placed in the harmony memory unless its objective value is less then the objective value of the worst harmony. Because of this, we denote the new harmony $\mathbf{x}_{new}$. In our example each harmony consists of two notes, i.e. an x and an y value. For each note we first have to decide whether it should be based on a previous note or not. If a random value between zero and one is smaller than $HMCR$ the new note will be based on a previous note. If not, a random note within variable bounds will be produced.

We create each note individually. We will start with the first note in the harmony $\mathbf{x}_{new,1}$, i.e. the x-value. In our example $HMCR = 0.8$. The produced random value turned out to be larger than 0.8. This means that a random x-value will be produced. In our case we got $\mathbf{x}_{new,1} = lb + (ub - lb) \times rand = 10 \times rand = 4.35$.

To create the y-value we use the same approach. This time the random value turned out to be smaller than 0.8. This means that the y-value will be based on a y-value from a harmony stored in the harmony memory. If a random value is larger than PAR, $\mathbf{x}_{new,2}$ will be a slight alteration of a randomly chosen y-value in HM. If the random value is smaller than PAR, $\mathbf{x}_{new,2}$ will be an exact copy of a randomly chosen y-value in HM. In this example the random value was smaller than PAR. Thus a y-value from HM will be chosen randomly. The y-value chosen was the y-value of the fourth harmony, i.e. $\mathbf{x}_{4,2}$. The deviation from $\mathbf{x}_{4,2}$ is partly decided from FW which is set to y-value upper bound minus y-value lower bound all divided by a houndred. I.e. $FW = (10-0)/100 = 0.1$. $\mathbf{x}_{new,2}$ will thus be given a randomly chosen value between $\mathbf{x}_{4,2} - 0.1$ and $\mathbf{x}_{4,2} + 0.1$, i.e. a random value in the range $[7.25, 7.45]$ In our example we got $\mathbf{x}_{new,2} = 7.35$. The new harmony is thus,

The two new notes form our new harmony, $\mathbf{x}_{new} = [4.35, 7.35]$. Now it's time to find the quality of the new harmony, i.e. the objective value ϵ_{new}, of our new solution. By using the objective function to evaluate the new harmony we get $\epsilon_{new} = f(\mathbf{x}_{new}) = 6.37$. Since $\epsilon_{new} < \epsilon_{worst}$, $\mathbf{x}_{new}$ will take the place of $\mathbf{x}_7$ in HM. An updated version of HM is presented in Table 2.7.

x	**y**	ϵ
1.49	6.75	0.82
9.94	8.45	2.81
6.44	6.20	-2.47
2.91	7.37	3.19
0.54	8.83	-6.88
3.22	3.78	-4.59
2.91	**7.35**	**6.37**
2.28	1.55	-0.60

Table 2.7: Harmony memory after one improvisation.

The next step is to update $\mathbf{x}_{best}$ and $\mathbf{x}_{worst}$ to prepare for the next improvisation.

2.5.5 Check for Convergence

After each improvisation HS checks for convergence. If $\mathbf{x}_{best}$ is below the stopping criteria or $MaxImp$ is reached, the algorithm is terminated and $\mathbf{x}_{best}$ is returned. New harmonies are improvised until a stopping criteria is met. In our example the minimum $\epsilon = f(0.9039, 0.8668) = -18.5547$ was found after 25 generations.

2.6 Downhill Simplex

The downhill simplex (DS) method was developed in the mid-1960s, and is a local optimizer that doesn't require the calculation of derivatives [10]. The optimization makes use of a simplex, which is the most elementary geometrical figure that can be created in dimension D. If $D = 2$, the simplex is a triangle, and if $D = 3$, the simplex is a tetrahedron. In other words, the simplex has $D + 1$ vertices and thus points in the solution space. The idea of DS is to move the simplex so it surrounds the minimum, and then to contract the simplex around the minimum

until an acceptable error is reached. The user specifies a starting point, P_0. D points are then created by the formula

$$P_i = P_0 + c_s e_i \tag{2.14}$$

where c_s is a scaling constant and e_i are one of $D+1$ unit vectors. After the initial simplex is created, the minimum is found through four operations: reflection, expansion, contraction and shrinkage. These operations are illustrated in Figure 2.7, where the simplex is minimizing a problem with $D = 2$. After the initial simplex is created, the objective value of each corner in the simplex is calculated. Then a new point, P^* is created by reflecting the point with the lowest objective value, P_n, through the centroid of the simplex, $\overline{P}$. This is shown mathematically in equation (2.15)

$$P^* = \overline{P} + \alpha(\overline{P} - P_n) \tag{2.15}$$

α is a reflection coefficient that controls the length of the reflection. After the reflection, the objective value of the new point is calculated. If the objective value of P^* is lower than the objective value of P_n, the move of the reflection was in the right direction. An expansion is thus made in the same direction as the reflection, and a new point P^{**} is created.

$$P^{**} = \overline{P} + \gamma(P^* - \overline{P}) \tag{2.16}$$

In equation (2.16) γ is an expansion coefficient that controls the size of the expansion. The objective value of the point generated by expansion is thus calculated. If the objective value of P^{**} is smaller than the objective value of P^* the vertices P^{**} is kept and P^* is deleted. After the expansion the step, the centroid for the new simplex is calculated and the algorithm jumps back to the reflection operation.

If $f(P^*) > f(P_n)$ this indicates that the move done by the reflection was either in the wrong direction or that the magnitude of the reflection was to big. P^* is then discarded and a contraction operation is performed.

$$P^* = \overline{P} + \beta(P_n - \overline{P}) \tag{2.17}$$

β is a contraction coefficient that controls the amount of contraction.

If $f(P^*) > f(P_n)$ the shrinkage operation is performed. The shrinkage operation implies keeping P_n but generating new points for all the D other points. This is done by the following formula:

$$P_i = \frac{(P_i + P_n)}{2} \tag{2.18}$$

where P_i indicates point number i and P_n as before is the point in the simplex with the lowest objective value.

For each iteration, a new simplex is generated. New iterations are performed until a maximum number of function calls are made, or the difference between the two points with highest and lowest objective are smaller than a predefined limit.

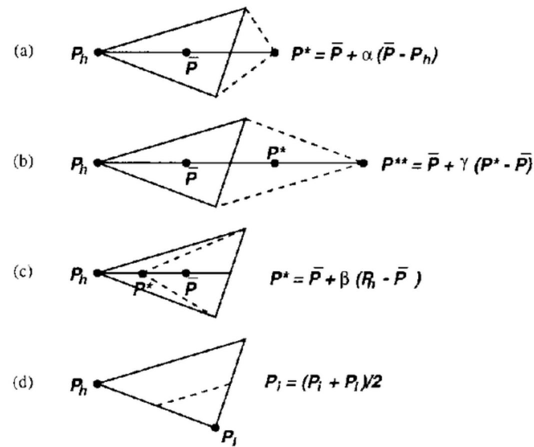

Figure 2.7: Reflection (a), expansion (b), contraction (c) and shrinkage (d) performed on a simplex solving a minimization problem with two variables. Figure reprinted from http://neural.cs.nthu.edu.tw/ [18].

CHAPTER 3

ESTIMATION OF GEOACOUSTIC PARAMETERS

The optimization methods presented in the previous chapter will be tested on an optimization problem taken from the field of geoacoustic inversion. This chapter will give a brief presentation of the problem.

The goal of the optimization is to estimate the velocities of upper oceanic crust based on synthetic bottom reflection loss as a function of grazing angle data. In real life this data is recorded.

3.1 Geoacoustic Parameters

The parameters we want to estimate are located in the sediment ($L1$) and basalt ($L2$) layers of the oceanic crust. There are a total of 11 geoacoustic parameters that is to be estimated. This means that $D = 11$. The compressional (P) and shear (S) wave velocities in $L1$ and $L2$ are denoted by V_{p1}, V_{s1}, V_{p2} and V_{s2} respectively. The compressional and shear wave attenuations in $L1$ and $L2$ are denoted by α_{p1}, α_{s1}, α_{p2} and α_{s2}. In addition to this the densities of $L1$ and $L2$, which are denoted by ρ_1 and ρ_2, are estimated. The final parameter is the sediment thickness and is denoted by H. These parameters are illustrated in Figure 3.1. Research has concluded that the compressional and shear wave velocities of the basalt in addition to sediment thickness are highly sensitive parameters in the inversion [4]. This means that it is possible to estimate accurate estimations of these parameters

through inversion. Other parameters are less sensitive, and are thus more difficult to estimate.

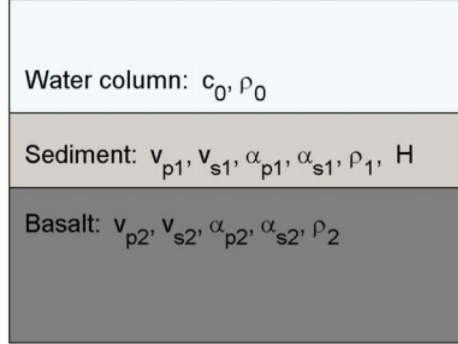

FIGURE 3.1: Water column, sediment and basalt layers with their respective parameters. Image is copied with the courtesy of [4].

3.2 Formulating the Inversion as an Optimization Problem

The reflection coefficients gathered through recording, can also be calculated by physical formulas, where the geoacoustic parameters we want to estimate are the input parameters. This means that for each grazing angle, $\theta_i \in [0\,^\circ\ 90\,^\circ]$, the plane wave reflection coefficient is a function of the geoacoustic parameters.

$$R = f(V_{p1}, V_{s1}, V_{p2}, V_{s2}, \alpha_{p1}, \alpha_{s1}, \alpha_{p2}, \alpha_{s2}, \rho_1, \rho_2, H, \theta_i) \tag{3.1}$$

For more detailed information about the reflection coefficient formula see [3] and [4]. Estimation is done by choosing the geoacoustic parameter values that minimize the euclidean distance between the rsynthesized and the calculated reflection coefficients as a function of grazing angle. The optimization error is thus a function of the geoacoustic parameters,

$$\epsilon = f(V_{p1}, V_{s1}, V_{p2}, V_{s2}, \alpha_{p1}, \alpha_{s1}, \alpha_{p2}, \alpha_{s2}, \rho_1, \rho_2, H) \tag{3.2}$$

and can be found by

$$\epsilon = \sqrt{(R_{d1} - R_{f1})^2 + (R_{d2} - R_{f2})^2 + ... + (R_{dN} - R_{fN})^2} \tag{3.3}$$

N is the total number of grazing angles, R_{di} represents the the recorded reflection coefficient from grazing angle number i and R_{fi} represents the reflection coefficient calculated from equation (3.1) for grazing angle number i. More compactly this can be written as

$$\epsilon(V_{p1}, V_{s1}, V_{p2}, V_{s2}, \alpha_{p1}, \alpha_{s1}, \alpha_{p2}, \alpha_{s2}, \rho_1, \rho_2, H) = \sqrt{\sum_{i=1}^{N}(R_{di} - R_{fi})^2} \tag{3.4}$$

The optimal solution, ϵ^* is found by minimizing Equation 3.4

$$\epsilon^* = \min_{V_{p1}, V_{s1}, V_{p2}, V_{s2}, \alpha_{p1}, \alpha_{s1}, \alpha_{p2}, \alpha_{s2}, \rho_1, \rho_2, H} [\quad \epsilon \quad] \tag{3.5}$$

The minimizing process for a synthetic example is illustrated in Figure 3.2. The blue lines shows the synthetic reflection coefficients, as a function of grazing angle. The red line shows the reflection coefficients gained by optimization. Optimization after 200, 500 and 1000 function calls using ASHS, is shown at the top, middle and bottom plots respectively.

The synthetic data is generated by inserting predefined values for the 11 optimization variables into equation (3.1) for all N values of θ. These values as well as the bound limits for all of the variables are given in Table 3.1. The bounds are set with the help of the projects supervisor.

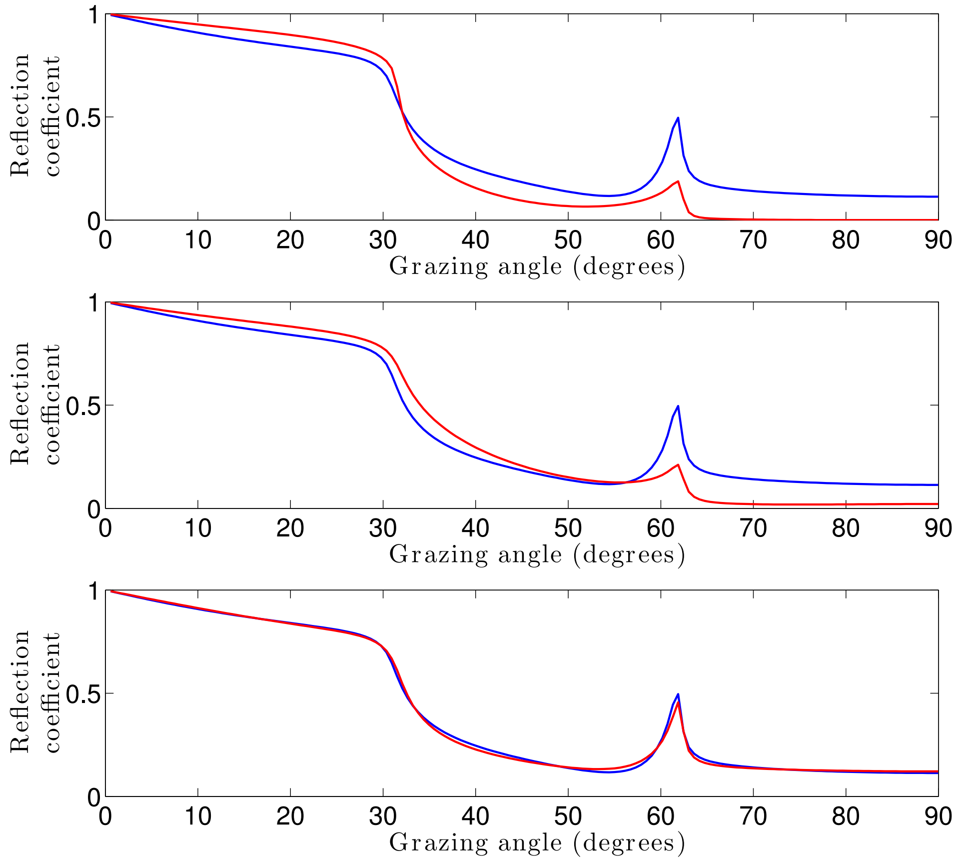

FIGURE 3.2: Optimization process by minimizing euclidean distance. Blue line represents synthetic reflection coefficients and red line represents reflection coefficients generated through optimization after 200 (top), 500 (middle) and 1000 (top) function evaluations.

Variable	True Value	Bound Min	Bound Max
V_{p1} (m/s)	1700	1000	2200
V_{p2} (m/s)	3200	2500	4000
V_{s1} (m/s)	300	100	600
V_{s2} (m/s)	1750	1200	2000
α_{p1} (dB/λ)	0.5	0	1
α_{p2} (dB/λ)	0.5	0	1
α_{s1} (dB/λ)	0.5	0	1
α_{s2} (dB/λ)	0.5	0	1
ρ_1 (kg/m^3)	1800	1200	2100
ρ_2 (kg/m^3)	2500	2000	3000
H (m)	40	5	60

TABLE 3.1: True value, minimum bound and maximum bounds for all the parameters.

CHAPTER 4

IMPLEMENTATION

4.1 Implementing Global Optimization Algorithms

All algorithms used in this project are implemented in MATLAB. The algorithms DE, HS and ABC are modifications of programs published by the respective algorithms authors in [22], [8] and [15] respectively. This ensures that the algorithms work as they are intended, and will thus make the comparison of the algorithms as fair as possible. GA is a modified version of Program 3: Continuous Genetic Algorithm from [9]. Based on the research on the use of ABC on real-parameter optimization [14], the control parameters MR and SF were implemented into the original ABC [15].

4.2 Creating Hybrid Algorithms

Global optimizers are great for searching the entire solution space. Their way of perturbing existing solutions increases the chances of finding a global optimum [9]. This is however often on the expense of fast convergence. Local optimizers on the other hand, like the DS, are powerful local decent algorithms, with fast convergence. The downside of the local optimizers is that they are easily entrapped in local optima, and are extremely sensitive to the initial starting point [12]. When combining a local and a global optimizer in a hybrid algorithm, the goal is to use

the strengths of both optimizers, while covering up for each other's weaknesses. The goal is to use the global optimizer's ability to explore, and the local optimizers ability to exploit. The final hybrid should both be able to search the entire solution space, and have a fast convergence.

There are many ways of combining global with local optimizers. The hybrids can roughly be divided into pipeline hybrids and hybrids where the local optimizer work as an additional operator inside the global optimizer [1]. In a pipeline hybrid, the two optimizers have a sequential workflow. Global searches based on the information stored in the entire population is performed before a given number of solutions is passed to the local optimizer for further improvement. These improvements can then be sent back to the global optimizer. The additional-operator-hybrid includes local search as if it was a part of the global optimizer. Due to the results of [1], a pipeline hybrid is used in this project.

Even pipeline hybrids can be implemented in several ways. The variations include the number of optimization iterations performed by each optimizer before the other one takes over, the number of starting points sent to the local optimizer, i.e. the number of parallel local optimizations, and the number of local optima passed back to the global optimizer. Some, [11] [1], initiates local search whenever the global search has found a new best solution. Others [25] claim that initiating local search after multiple successive global search iterations in a row damages the process of the global search and should be avoided. They initiate a local search whenever the local search is unable to improve the best solution after a given number of iterations. Others [17] initiate local optimization after every global search iteration . This project presents a hybrid where the local search is initiated when the global search has made N_{global} function calls. N_{global} is a parameter that can be adjusted by the user to fit the problem at hand.

The flow of the hybrid algorithms presented in this thesis is shown in Figure 4.1. The workflow allows the use of any global optimization method. Optimization starts with standard initialization; control parameters are selected and the variables and objective function are defined. An initial population of N_{pop} members with D variables are then created by randomly picking points from the solution space. The next step is the global search. This includes mutation, crossover and selection for DE, improvising a new harmony and selection for HS and employed bee phase, onlooking bee phase and scout bee phase for the ABC. The global search is run until it has used N_{global} function evaluations. The best solution is

thus used as a starting point for DS. DS using N_{local} function evaluations is then performed. Former projects [17] have had success with an increasing N_{local} for each iteration. This has been implemented to this project's hybrids by adding another user-defined control parameter: $L_{incfact}$. $L_{incfact}$ is a factor defining how much N_{local} increases on each iteration. If $L_{incfact} = 1.2$, N_{local} will increase by 20% every time the hybrid goes from global to local search. This allows for more thorough local searches as the hybrid is getting closer to the global optimum. $L_{incfact}$ makes the hybrids adaptive as a function of time. Because of this, the hybrids are given the prefix AS. The hybrids are thus named adaptive simplex differential evolution (ASDE), adaptive simplex artificial bee colony (ASABC) and adaptive simplex harmony search (ASHS). Some hybrids used i previous research [11][6] have included a perturbation step inside of the downhill simplex. The perturbation size in these hybrids has decreased as a function of time in order to focus on exploitation rather than exploration towards the end of the optimization process. It should be noted that these hybrids have the *AS* prefix because of the adaptive perturbation size. The choice of excluding the perturbation step was made in order to make it easier to compare the hybrid versions of the different global optimizers. One hybrid iteration is defined as N_{global} function calls by the global optimizer and N_{local} function calls by DS.

Figure 4.1 shows a stop criteria check after each DS. This stop-criteria-check is in fact performed for every iteration of both the global and the local search. This means that the hybrid algorithm will terminate immediately after the objective value of it's best solution is below the stop criteria, ϵ_{tol}. If the best solution found by the DS is better than the worst solution in the population, it takes the place of the worst population member in the population.

N_{global} could have been a value for number of generations or full algorithm iterations rather than function calls. Function calls were chosen in order to have a generic hybrid design that could be used in all three algorithms. Note that the counting of N_{global} calls starts after the initialization. This means that the population chosen randomly during initialization is evaluated before the counting of N_{global} starts.

The hybrid continues to switch between the local and global optimizers like this until a stop criteria is met.

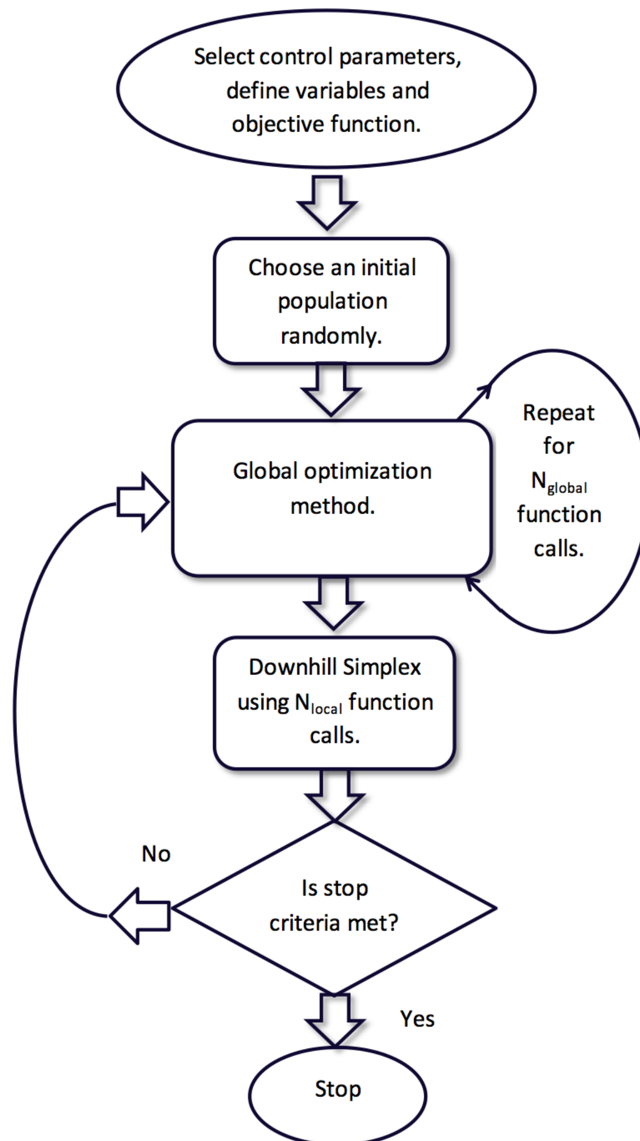

FIGURE 4.1: Flowchart of the hybrid optimizer algorithm.

4.3 Finding the Control Parameters of the Global Optimizers

The control parameters of each algorithm have a great impact on overall performance. A great algorithm with improper control parameter values can easily be outperformed by a far less effective algorithm with finely tuned control parameter values. Finding acceptable control parameter values is not a linear problem. A low N_{pop} might for instance be good in some control parameter combinations, and bad in others. Because of this, good control parameter settings should be found by testing multiple combinations of control parameters values rather than variating the value of one control parameter at the time. Performing control parameter tests also result in information about the sensitivity of each control parameter.

A substantial amount of testing was done in order to find good control parameter values for all of the algorithms. Acceptable control parameters values were found by counting the number of objective function calls needed to reach a predefined optimization error limit, ϵ_{tol}, under different control parameter value combinations. The geoacoustic problem presented in Chapter 3 was used. Due to the random elements in the global optimizers used in this project, two runs are never identical. Thus each setting was tested five times to prevent a control parameter setting from being based on a lucky run. The average of these five runs were used to choose the most effective settings. If a run had not met ϵ_{tol} before 50000 iterations, the run was terminated and labeled as non-converging. The average number of function calls was calculated from only the converging runs. Some control parameter settings are very unstable in the way that they find an optimum very fast in some runs, and don't manage to find the optimum at all in other runs. Since the average number of function calls is calculated from only the converging runs, the unstable control parameter settings often have a low average number of function calls, but a high rate of non-converging runs. Because of this, only combinations with no non-converging runs were chosen in the final settings.

Since a large number of control parameter combinations have been tested, each control parameter value has been tested in many combinations. This gives us a chance to find the average number of function calls used in all the combinations where the given control parameter value is included. By doing this for all values of the given control parameter, we find the average number of function calls as a function of the respective control parameter. This resembles a linear-problem

solution to finding the best control parameter values and will thus not necessarily give the best control parameter values. It does however give us information about how the control parameters in general affects output. For instance, let's say we find the average number of function calls used in all the combinations tested for GA where $N_{pop} = 10$. Further we repeat this process for all combinations that include $N_{pop} = 15$. If this process is repeated for all the tested N_{pop} values, the results will give information about both the sensitivity of N_{pop}, and an indication of what control parameter values for N_{pop} that generally leads to good solutions independent on the values of the other control parameters. This information is presented for all control parameters for all algorithms in the following sections. It should be emphasized that since finding control parameters is a non-linear problem a control parameter value might be a part of the best control parameter combination, even though the average of all combinations where that control parameter value is included has worse performance than the average of all combinations of another value of the same control parameter.

The range of the control parameter values that was tested has been decided by looking at previous projects and by looking at suggestions from the respective algorithm's authors.

4.3.1 GA

The GA was tested with 9 different population sizes from 10 to 270 with an increment factor of 1.5. The selection rate was varied between 0.5, 0.6, 0.7 and 0.8. The mutation rate was varied between 0.1, 0.15, 0.20, 0.30 and 0.4. Every combination of these values were tested five times, resulting in a total of 900 runs.

The combination resulting in the lowest number of function calls averaged over five runs with equal settings was
$N_{pop} = 10$
$X_{rate} = 0.5$
$\mu = 0.2$
This combination made an average of 1460 function calls in five runs.

Tables 4.1, 4.2 and 4.3 show how the number of function calls needed to reach ϵ_{tol} varied as a function of population size, selection rate and mutation rate respectively. It can be seen that a low population size is suited for this problem. This

can be seen by looking at both the average number of function calls used, and the percentage of non-converging runs. The best results, both in terms of number of function calls and non-converging runs, were found when the selection rate was low. The same goes for mutation. It can be seen that the best performing combination has different N_{pop} and μ values than the best values for N_{pop} and μ when averaged over all possible combinations. This shows that finding good control parameter values is not a linear problem. Because of the random nature of the metaheuristics, this could also be a coincidence. However it does make sense that lower population sizes work well with higher mutation rates, as fewer population members increases the probability of getting stuck in local minima. This problem can in some cases be solved by a higher mutation rate, which helps to slow down convergence and search a wider area of the solution space.

For the population size, the highest average number of function calls was 74.43% higher than the lowest. Similarly the difference was 31.77% for the selection rate and 55.53% for the mutation rate. This shows us that the population size is the most sensitive control parameter for GA solving this optimization problem.

N_{pop}	**Avg. number of function calls** (10^4)	Non-converging runs
10	1.7623	40%
15	1.7100	38%
23	1.4538	58%
35	2.0017	65%
54	1.9676	53%
80	2.2574	69%
120	2.2067	71%
180	2.2616	68%
270	2.5358	68%

Table 4.1: GA, Average optimization error as a function of population size, N_{pop}.

X_{rate}	**Avg. number of function calls** (10^4)	Non-converging runs
0.5	1.6450	47%
0.6	1.8976	54%
0.7	2.1676	60%
0.8	2.1123	73%

Table 4.2: GA, Average optimization error as a function of the selection rate, X_{rate}.

μ	**Avg. number of function calls** (10^4)	Non-converging runs
0.1	1.5184	35%
0.15	1.6824	51%
0.20	2.2181	44%
0.30	2.3616	75%
0.40	2.2608	88%

Table 4.3: GA, Average optimization error as a function of mutation rate, μ.

4.3.2 DE

DE was tested with 9 different values of N_{pop} from 30 to 770 with an increment of 1.5. Previous literature suggests setting $N_{pop} = D \times 10$ [21] and $N_{pop} = D \times 20$ [11]. This shows that the population size often is set higher in DE than in GA and ABC. This is the reason for testing different values for N_{pop} for the DE than for GA, HS and ABC. DE was further tested with four values of CR from 0.7 to 1, and six values of F_{weight} from 0.5 to 1. In total 1080 runs were made.

The following combination of control-parameters proved to be most efficient.

$N_{pop} = 30$
$CR = 1$
$F_{weight} = 0.5$

Tables 4.4, 4.5 and 4.6 shows how the number of function calls varied as a function of N_{pop}, CR and F_{weight} respectively. It can be seen that performance is highest for low N_{pop}, high CR and low F_{weight}.

N_{pop} is the most sensitive of the three with a difference of 604% between the lowest and highest value. This might be a consequence of the fact that N_{pop} has been

tested for a very wide range of values. CR seems to be the least sensitive control parameter.

Unlike the other three algorithms, the values in the best control parameter combination is the same as the best values for each control parameter when averaged over all combinations. This implies that finding the control parameter values for DE could be solved as a linear problem.

N_{pop}	**Avg. number of function calls** (10^4)	Non-converging runs
30	0.6374	1%
45	0.9896	0%
68	1.4995	0%
102	1.9990	4%
152	2.7427	17%
228	3.1050	28%
342	3.3302	49%
514	4.0266	71%
770	3.8507	85%

Table 4.4: DE, Average number of function calls as a function of population size, N_{pop}.

CR	**Avg. number of function calls** (10^4)	Non-converging runs
0.7	0.8403	33%
0.8	0.7726	30%
0.9	0.5371	27%
1	0.3997	21%

Table 4.5: DE, Average number of function calls as a function of the crossover constant, CR.

F_{weight}	**Avg. number of function calls**(10^4)	Non-converging runs
0.5	1.7075	6%
0.6	1.8882	15%
0.7	2.2433	22%
0.8	2.4333	33%
0.9	2.4709	42%
1	2.7028	53%

Table 4.6: DE, Average number of function calls as a function of F_{weight}.

4.3.3 ABC

The ABC was tested with the same 9 different population sizes as GA. Since the number of employed bees equals to half the population size, the tested population sizes are rounded up to the next even number. The modification rate was varied between 0.1 and 0.4 with an increment of 0.05. SF_{change} was varied between 0.85, 0.95, 0.995 and 0.9995. Every combination of these values were tested five times, resulting in a total of 1260 runs.

The combination resulting in the lowest number of function calls averaged over five runs with equal settings was
$N_{pop} = 16$
$MR = 0.1$
$SF_{change} = 0.995$

This combination made an average of 4715 function calls in five runs.

Table 4.7, 4.8 and 4.9 shows how the number of function calls varied as a function of population size, modification rate and SF_{change} respectively. As in GA, runs with a low population size performed best. The performance of the ABC was much better for $MR = 0.1$ than all the other values. Based on the results, SF_{change} should be given a value close to 1.

It can be seen that when averaged over all combinations, $N_{pop} = 10$ had the lowest average number of function calls. This is however on the expense of a high rate of non-averaging runs. When using a very small population size, it becomes difficult to explore the entire solution space. This makes the quality of the initial food sources very important. If the initial food sources are are close to an optimum, the bees might find the solution very fast. If the initial food sources are far from an optimum however, they might have to search for a long time before they find it. This might explain why $N_{pop} = 10$ was so unstable in the control parameter tests. The high non-converging rate might also be the reason for $N_{pop} = 10$ not being in the best combination, as no combinations with $N_{pop} = 10$ had five good converging runs.

The population size was even more sensitive than for the GA. The highest number of function calls was 76.18% higher than the lowest. It was however not as sensitive as the modification rate, which had a difference of 87.92. SF_{change} had a difference

between the lowest and the highest of 52.02%. This makes it the least sensitive of the control parameters tested, even though it is quite sensitive.

N_{pop}	**Avg. number of function calls** (10^4)	Non-converging runs
10	1.6462	41%
16	2.0331	32%
24	1.6882	37%
36	2.0730	39%
54	2.2310	42%
80	2.1696	35%
120	2.9002	42%
180	2.8810	45%
270	2.8489	60%

Table 4.7: ABC, Average optimization error as a function of population size, N_{pop}.

MR	**Avg. number of function calls** (10^4)	Non-converging runs
0.1	1.6496	21%
0.15	2.1101	18%
0.2	2.0861	16%
0.25	2.1956	33%
0.3	2.5798	50%
0.35	2.6676	69%
0.4	3.0999	85%

Table 4.8: ABC, Average optimization error as a function of modification rate, MR.

SF_{change}	**Avg. number of function calls** (10^4)	Non-converging runs
0.85	2.8178	67%
0.95	2.5631	47%
0.995	1.8530	32%
0.9995	1.9719	20%

Table 4.9: ABC, Average optimization error as a function of SF_{change}.

4.3.4 HS

To find acceptable values for the control parameters of the harmony search, the algorithm was tested with eight values for HMS, three values for $HMCR$, five values for PAR and four values for FW. This resulted in a total of 480 combinations, each run five times.

The combination resulting in the lowest number of function calls averaged over five runs with equal settings was

$$HMS = 28$$
$$HMCR = 0.9$$
$$PAR = 0.4$$
$$FW = \frac{ub-lb}{100}$$

This combination made an average of 1731 function calls in five runs.

Tables 4.10, 4.11, 4.12 and 4.13 show how the number of function calls varied as a function of HMS, $HMCR$, PAR and FW respectively. It can be seen that medium valued HMS and high valued $HMCR$ gave the best results. The best value for the FW was definitely $\frac{ub-lb}{100}$.

It can be noticed that PAR by far is the least sensitive of HS's control parameters. This shows that finding an optimal value for PAR might not be as important for the performance as for the other control parameters. FW however turned out to be highly sensitive, where the highest average number of function calls was 111% higher than the lowest. The tests thus tells us that an optimal value for this control parameter is vital for the performance of the HS.

The values of PAR and HMS found in the best control parameter combination, is not the same as best value averaged over all combinations as seen in Tables 4.12 and 4.10. The difference for PAR can be explained with the fact that is not very sensitive, but another theory is also a probability. Since PAR decides how often a value found in the harmony memory should be perturbed, it decides the frequency of perturbation. The harmony memory closely resembles the population size of GA, DE, and ABC. In the same way as discussed in previous sections, will a small harmony memory size make it hard to search the whole solution space.

This can be compensated for by increasing rate of perturbation, which can be done by increasing PAR. This might explain why a lower HMS value and a higher PAR value than in Tables 4.12 and 4.10 were found in the best combination. The benefit of a smaller harmony memory can be faster convergence.

HMS	**Avg. number of function calls** (10^4)	Non-converging runs
8	2.7369	19%
12	2.1555	10%
18	2.1855	16%
28	1.7372	10%
42	1.6594	8%
64	1.7589	7%
96	1.8243	7%
144	2.1511	7%

Table 4.10: HS, Average number of function calls as a function of harmony memory size, HMS.

HMCR	**Avg. number of function calls** (10^4)	Non-converging runs
0.7	2.4808	14%
0.8	1.9420	10%
0.9	1.6555	9%

Table 4.11: HS, Average number of function calls as a function of harmony memory considering rate, $HMCR$.

PAR	**Avg. number of function calls**(10^4)	Non-converging runs
0.1	1.8228	10%
0.2	2.0071	11%
0.3	1.9791	11%
0.4	1.9303	9%
0.5	2.3857	14%

Table 4.12: HS, Average number of function calls as a function of pitch adjusting rate, PAR.

FW	**Avg. number of function calls** (10^4)	Non-converging runs
$\frac{ub-lb}{10}$	2.4362	13%
$\frac{ub-lb}{100}$	1.2540	3%
$\frac{ub-lb}{1000}$	1.7631	9%
$\frac{ub-lb}{10000}$	2.6511	17%

Table 4.13: HS, Average number of function calls as a function of fret width, FW.

4.4 Finding the Control Parameters of the Hybrids

In addition to the control parameters of the global optimizers, the hybrids have three control parameters: N_{global}, N_{local} and $L_{incfact}$. These parameters control the amount of exploration and exploitation of the searches. A total of 300 different combinations have been tested five times each. Ideally, all of the hybrid control parameters, including the control parameters of the global optimizers, should be included in these tests. This would result in 300 times more combinations to test than for the global optimizers alone. Because of this, the global optimizer part of the hybrids use the same control parameters as found in the previous section.

The combination resulting in the lowest number of function calls averaged over five runs with equal settings are presented in Table 4.14.

	ASDE	**ASABC**	**ASHS**
N_{global}	76	171	171
N_{local}	380	169	254
L_{incfac}	0.2	0.2	0.2

Table 4.14: Best control parameter combinations for ASDE, ASABC and ASHS.

Since the hybrid control parameters are the same for all three algorithms, their average performance as a function of each control parameter can be plotted together. Figure 4.2 shows the average number of function calls as a function of N_{global} for all three algorithms. The results show that all three algorithms have the highest performance for mid-to-high values of N_{global}.

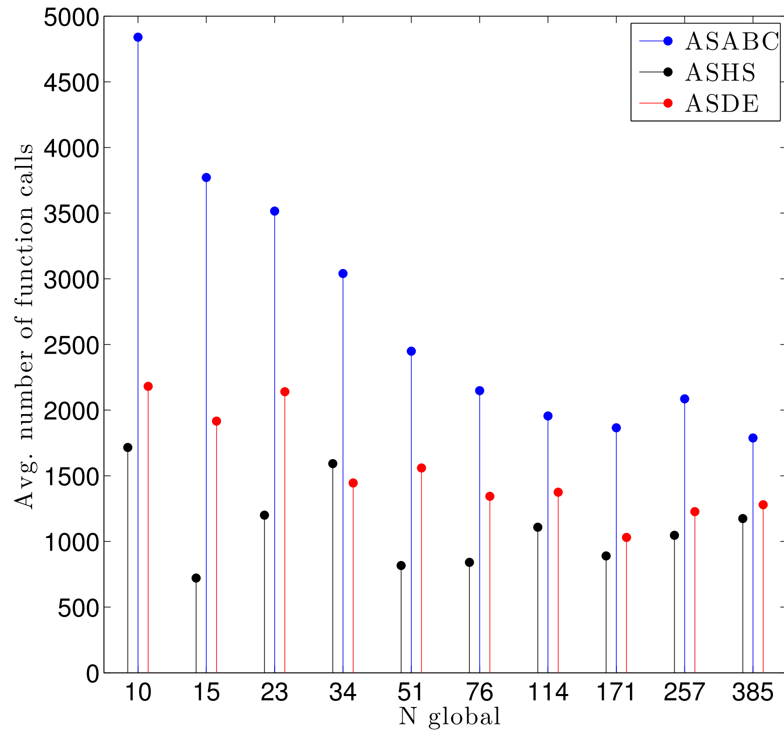

FIGURE 4.2: Avg. number of function runs as a function of N_{global}.

Figure 4.3 shows the average number of function calls as a function of N_{local}. Values between 50 and 380 with an increment factor of 1.5. The plot clearly shows that $N_{local} = 50$ doesn't give the DS enough function calls to fully descend the valley close to the starting point given by the global optimizer. It can be seen that a high-valued N_{local} gives the best results for all three algorithms.

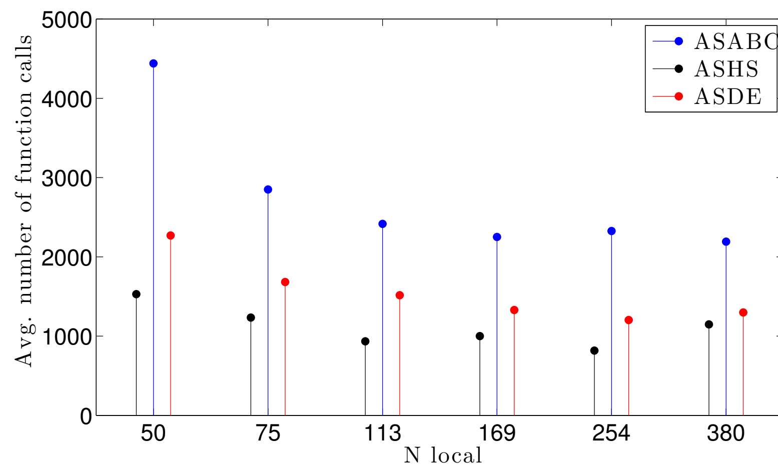

FIGURE 4.3: Avg. number of function calls as a function of N_{local}.

Figure 4.4 shows the average number of function calls as a function of $L_{incfact}$ for all three algorithms. It can be seen that this control parameter is of very low sensitivity. The deviation in average number of function calls between the different $L_{incfact}$ values is so small that little or no information can be extracted from the plot.

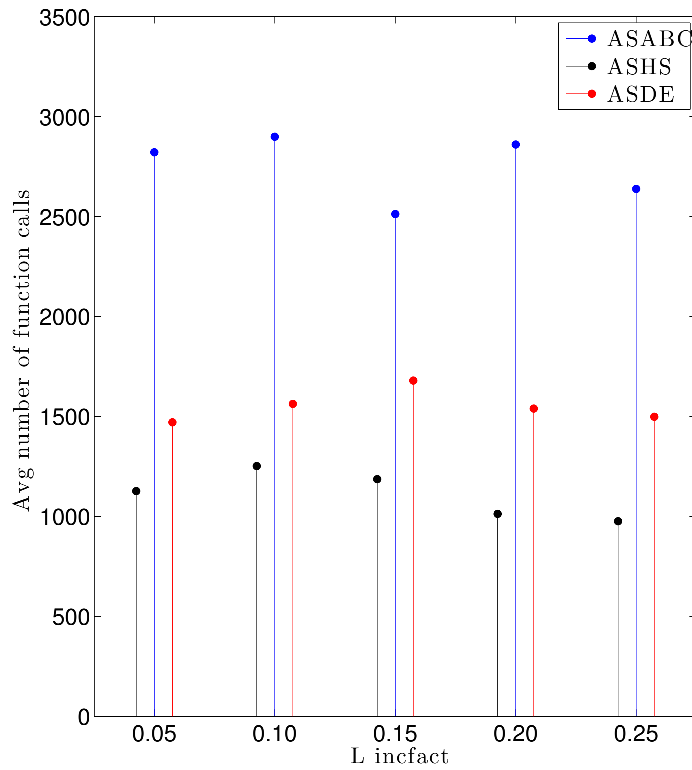

Figure 4.4: Avg. number of function calls as a function of $L_{incfact}$

The results in the figures are also presented in Tables 4.18, 4.19, 4.20, 4.21, 4.22, 4.23, 4.15, 4.19 and 4.17.

N_{global}	**Avg. number of function calls** (10^4)
10	0.2181
15	0.1916
23	0.2140
34	0.1445
51	0.1559
76	0.1344
114	0.1375
171	0.1030
257	0.1227
385	0.1279

Table 4.15: ASDE, Average number of function calls as a function of N_{global}.

N_{local}	**Avg. number of function calls** (10^4)
50	0.2269
75	0.1682
113	0.1516
169	0.1328
254	0.1203
380	0.1298

Table 4.16: ASDE, Average number of function calls as a function of N_{local}.

$L_{incfact}$	**Avg. number of function calls** (10^4)
0.05	0.1470
0.10	0.1562
0.15	0.1679
0.20	0.1539
0.25	0.1497

Table 4.17: ASDE, Average number of function calls as a function of $L_{incfact}$.

N_{global}	**Avg. number of function calls** (10^4)
10	0.4840
15	0.3771
23	0.3515
34	0.3039
51	0.2448
76	0.2148
114	0.1955
171	0.1865
257	0.2085
385	0.1787

TABLE 4.18: ASABC, Average number of function calls as a function of N_{global}.

N_{local}	**Avg. number of function calls** (10^4)
50	0.4440
75	0.2849
113	0.2415
169	0.2250
254	0.2325
380	0.2191

TABLE 4.19: ASABC, Average number of function calls as a function of N_{local}.

$L_{incfact}$	**Avg. number of function calls** (10^4)
0.05	0.2820
0.10	0.2898
0.15	0.2511
0.20	0.2859
0.25	0.2637

TABLE 4.20: ASABC, Average number of function calls as a function of $L_{incfact}$.

N_{global}	**Avg. number of function calls** (10^4)
10	0.1715
15	0.0720
23	0.1199
34	0.1592
51	0.0816
76	0.0840
114	0.1108
171	0.0889
257	0.1046
385	0.1174

TABLE 4.21: ASHS, Average number of function calls as a function of N_{global}.

N_{local}	**Avg. number of function calls** (10^4)
50	0.3998
75	0.1962
113	0.1105
169	0.1850
254	0.0507
380	0.0866

TABLE 4.22: ASHS, Average number of function calls as a function of N_{local}.

$L_{incfact}$	**Avg. number of function calls** (10^4)
0.05	0.1126
0.10	0.1251
0.15	0.1185
0.20	0.1012
0.25	0.0975

TABLE 4.23: ASHS, Average number of function calls as a function of $L_{incfact}$.

CHAPTER 5

RESULTS AND DISCUSSION

Algorithms GA, DE, ASDE, ABC, ASABC, HS and ASHS have been tested on both a standard benchmark problem and a problem of geoacoustic inversion. Only the evolutionary algorithm with the best results were implemented as a hybrid algorithm, meaning that no hybrid version of GA has been implemented. Tests were performed in MATLAB on a laptop computer. This chapter presents the results of these tests as well as a discussions regarding the matters presented in Chapter 1. Both results on algorithm speeds and accuracy will be presented. First of all however, the variable sensitivities will be attended.

5.1 Variable Sensitivities

The variable sensitivities gives an idea of which variables that have the biggest impeact on ϵ. It should not be confused with the sensitivity of the control parameters. To get an impression of how sensitive the different variables are, the optimization error, ϵ, was plotted as a function of each of the variables during optimization. During these sensitivity tests, ASHS with $\epsilon_{tol} = 0.05$ was used. This is presented in Figure 5.1. The blue dots represent values found by HS, and the red dots show the values found by DS. It can be seen that some variables can have a wide range of values even at low ϵ values. This indicates that these variables are less sensitive, and are thus more difficult to estimate. Based on this plot the most

sensitive variables are V_{s1}, V_{s2} and H. The rest of the variables seem to be less sensitive. This should be kept in mind especially during accuracy testing.

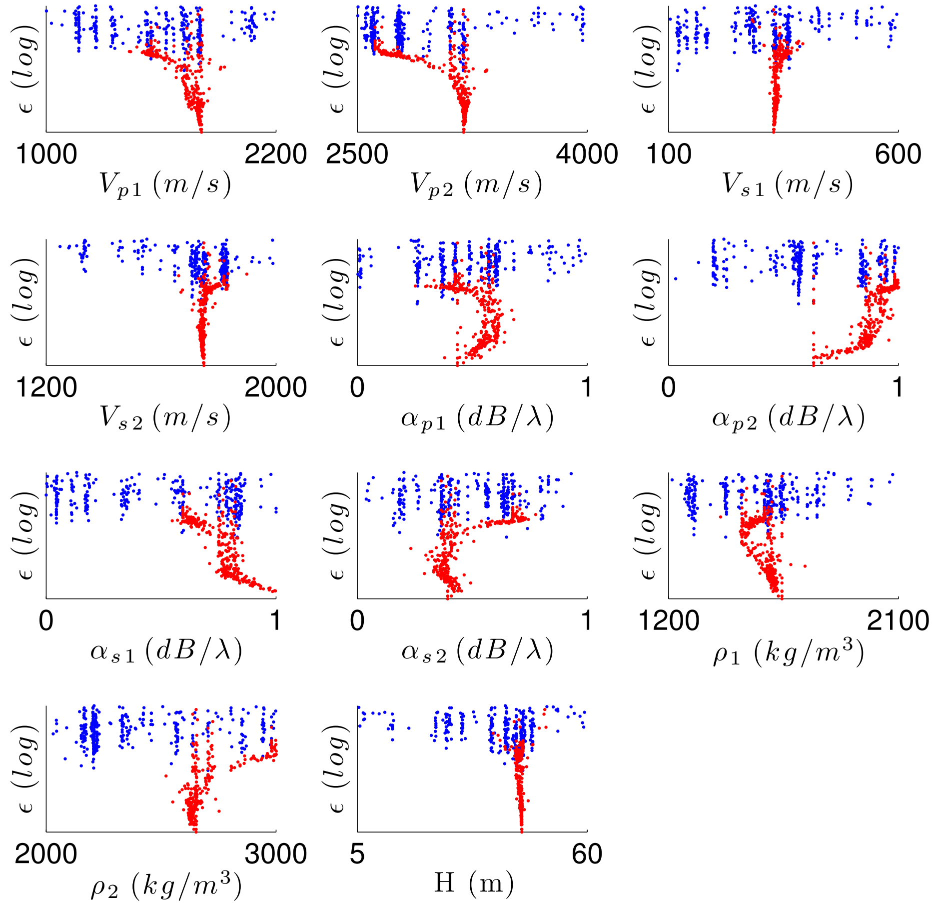

FIGURE 5.1: ϵ plotted as a function of each of the variables during a run of ASHS width $\epsilon_{tol} = 0.05$. Blue dots are function calls made by HS and the red dots are function calls made by DS. The plots gives an impression of each of the variables sensitivity.

5.2 Testing Algorithm Speed

The most time-consuming part of most metaheuristic algorithms is calls to the objective function. The algorithms and objective functions used in this project are no exceptions. Because of this, the algorithms are tested by counting the number of function calls needed to reach a given tolerance limit, ϵ_{tol}. Measuring

the number of function calls used is a way of measuring the computational cost. The results from the geoacoustic inversion problem will be presented after the results from the Rosenbrock problem.

5.2.1 Rosenbrock Function

5.2.1.1 Setup

The benchmark problem chosen for testing was the Rosenbrock function [19] given in equation (5.1).

$$f(x_1, x_2) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2 \quad x_1, x_2 \in [-2.048, 2.048] \tag{5.1}$$

The Rosenbrock function has its global optimum at $x_1 = x_2 = 0$. During the tests each algorithm was run ten times. The test measured the average number of function calls the algorithms needed to reach $\epsilon_{tol} = 0.025$. Runs where ϵ_{tol} was not reached before 7000 function calls were terminated and labeled as non-converging. The converging runs were used to calculate the average number of function calls needed to reach ϵ_{tol}. The control parameters found in Chapter 3 were used in these tests.

5.2.1.2 Results

Table 5.1 shows the results from the tests performed on the Rosenbrock function. These results make it very clear that the implementation of DS greatly increases the global optimizers performance. All three hybrids (ASDE, ASABC and ASHS) perform much better than their respective global optimizers. This is evident both when it comes to the average number of function calls needed to reach ϵ_{tol}, and in the percentage of non-converging runs.

Of the algorithms without adaptive simplex, DE has the fewest average function calls, and was thus the fastest. HS was slower, but much more stable, having no non-converging runs. Of the algorithms with adaptive simplex, the number of function calls needed was very similar. The results show that ASABC was the fastest, and ASHS was the slowest.

	Without AS		With AS	
	Avg.	Non-converging	Avg.	Non-converging
GA	26242	20%	NA	NA
DE	2425	20%	123	0%
ABC	43863	40%	117	0%
HS	6466	0%	130	0%

TABLE 5.1: Results from runs performed on the Rosenbrock function.

5.2.2 Estimation of Geoacoustic Parameters

5.2.2.1 Setup

The problem of geoacoustic inversion is presented in Chapter 3. The objective function is given in equation (3.4) and the variable bounds and correct values are given in Table 3.1.

The speed of each algorithm was tested by finding the average number of function calls needed to reach $\epsilon_{tol} = 0.15$. The value for ϵ_{tol} was found with the help of the project's supervisor, and results in an adequate estimation of the geoacoustic parameters. Each algorithm was run 200 times to eliminate the algorithm's deviation in performance caused by the use of random functions. Each run was terminated and labeled as non-converging if ϵ_{tol} had not been reached before 50000 function calls. The average was calculated from the converging runs. An identical approach was later used to test the three hybrid algorithms with $\epsilon_{tol} = 0.05$.

Originally the three hybrid algorithms were run with the control parameters found in Chapter 4. This resulted in poor performance for ASDE and ASABC with an average number of function calls of 1340 and 1587 respectively. Their performance was not even close to the performance of ASHS. Because of this, ASDE and ASABC were given the same adaptive simplex control parameters as ASHS ($N_{global} = 171$ and $N_{local} = 254$). This resulted in higher performance, and the settings were thus kept for the rest of the tests. Of the results presented in this thesis, only the tests performed on the Rosenbrock function had the adaptive simplex control parameters as found in Chapter 4.

5.2.2.2 Results

The results from the tests performed with $\epsilon_{tol} = 0.15$ are presented in Table 5.2. It can be seen that the average number of function calls greatly decreases with the use of adaptive simplex in addition to the global optimizers. HS has the fewest average amount of function calls both with and without adaptive simplex. ASHS and ASABC are the only algorithms without any non-converging runs.

	Without AS		With AS	
	Avg.	Non-converging	Avg.	Non-converging
GA	8329	13.5%	NA	NA
DE	2816	1.5%	946	1.5%
ABC	5694	1.5%	1526	0%
HS	2316	1%	784	0%

TABLE 5.2: Average results and rate of non-converging runs recorded over 200 runs for each algorithm. $\epsilon_{tol} = 0.15$.

Table 5.3 shows the results from tests performed with $\epsilon_{tol} = 0.05$. These test where only performed on the hybrid algorithms. The results show that ASHS has the fastest convergence. ASABC has the slowest convergence of the three, but it is the only one without non-converging runs.

	Avg.	Non-converging
ASDE	1645	1%
ASABC	2758	0%
ASHS	1576	1%

TABLE 5.3: Average results and rate of non-converging runs recorded over 200 runs for each algorithm. $\epsilon_{tol} = 0.05$.

5.3 Testing Algorithm Accuracy

5.3.0.3 Setup

In order to test the algorithms accuracy, the three hybrids were tested with no stop criteria, i.e. $\epsilon_{tol} = 0$, and a predefined number of function calls was set to 5000. The goal of the tests was to see which algorithm could produce the most accurate solution. Each algorithm was run 200 times.

5.3.0.4 Results

The average and minimum ϵ as well as the solution of the run with the lowest ϵ is given in Table 5.4. From these results it is apparent that ASHS is the most accurate of the three hybrids solving this inversion problem. ASHS both has the lowest average ϵ and by far the lowest minimum ϵ. The variable with the largest deviation from true value, has a deviation of 0.36%, which is far lower than for the two others. By looking at the best solution found by ASDE and ASABC, it can be seen that V_{s2},V_{p2} and h are the most sensitive parameters. This is consistent with previous work presented in [4]. The tests suggests that α_{p1} is the least sensitive variable.

	ASHS	ASDE	ASABC
Average value	2.4×10^-2	3.2×10^-2	3.9×10^-2
Minimum value	5.2×10^-5	3.1×10^-3	4.7×10^-3
Best solution	V_{p1}=1700.11 [0.00632%] V_{p2} = 3200.01 [0.00042%] V_{s1} = 299.98 [0.00623%] V_{s2} = 1749.99 [0.00057%] α_{p1} = 0.50 [0.02953%] α_{p2} = 0.5 [0.02234%] α_{s1}=0.5 [0.02139%] α_{s2}=0.500 [0.00491%] ρ_1=1800.03 [0.00184%] ρ_2=2499.79 [0.36154%] d=39.99[0.00383%]	V_{p1}=1701.59 [0.09343%] V_{p2} = 3201.17 [0.03657%] V_{s1} = 297.47[0.84322%] V_{s2} = 1750.10 [0.00570%] α_{p1} = 0.62[24.39530%] α_{p2} = 0.49[1.45341%] α_{s1}=0.42 [14.98868%] α_{s2}=0.51 [0.40182%] ρ_1=1825.74 [1.43025%] ρ_2=2508.98 [0.35932%] d=39.78[0.55376%]	V_{p1}=1.714.57 [0.85704%] V_{p2} = 3200.10 [0.00323%] V_{s1} = 301.06 [0.35336%] V_{s2} = 1750.17[0.00962%] α_{p1} = 0.61[21.26598%] α_{p2} =0.49[1.44251%] α_{s1}=0.51 [1.33595%] α_{s2}=0.49 [2.18565%] ρ_1=1793.49 [0.36%] ρ_2=2517.43[0.69756%] d=40.25[0.61912%]

TABLE 5.4: Avg. and min. value for 200 runs with $\epsilon_{tol} = 0$. The variable values of each algorithm's best solution is found in the last row.

5.4 Discussion

5.4.1 Improvement of Including Adaptive Simplex

A part of this project was to find out if the performance of HS and ABC would be improved if they were combined with a local optimizer. The tests clearly state that adding adaptive simplex to the global optimizers increases performance both in terms of speed and accuracy. Figure 5.2 shows the results from the tests on ASHS (blue) and ASDE (red) with $\epsilon_{tol} = 0.15$. For each of the 200 runs, the number of function calls have been plotted. Based on this figure it is difficult to extract much information other than the fact that ASDE has a higher frequency of bad performing runs than ASHS. In order to analyse the test results further, the data in Figure 5.2 have been sorted in ascending order in terms of the number of function calls. This is plotted in Figure 5.3. Based on this figure it can be seen that the number of function calls needed to reach ϵ_{tol} is divided into "steps". Let further *init* represent the number of function calls made during algorithm initialization. Given the control parameters found in Chapter 4, this equals to 16 (N_{pop}) for ASABC, 28 (HMS) for ASHS and 30 (N_{pop}) for ASDE. Since the tests where performed with $N_{global} = 171$ and $N_{local} = 254$, the function calls made in the intervals $\mathbb{Z}[171 + init, 425 + init]$, $\mathbb{Z}[596 + init, 850 + init]$ and so on, are function calls made by the DS part of the algorithms. Since almost all runs are terminated in these intervals it can be concluded that the tolerated solutions were found during or shortly after DS in almost all of the runs. Based on the results and this plot it can be concluded that the hybrid optimizers used in this thesis have higher performance than the global optimizersalone, when solving the geoacoustic inversion problem.

5.4.2 ASHS versus ASDE

The second and most important task of this project was to see if combinations of DS with relatively recent global optimizers could outperform ASDE. The results clearly shows that ASHS has higher performance than ASDE both in terms of speed and accuracy when solving the geoacoustic inversion problem. The same can not be said about ASABC, which had the worst performance of the three hybrids. This part of the discussion will because of this cover the comparison of

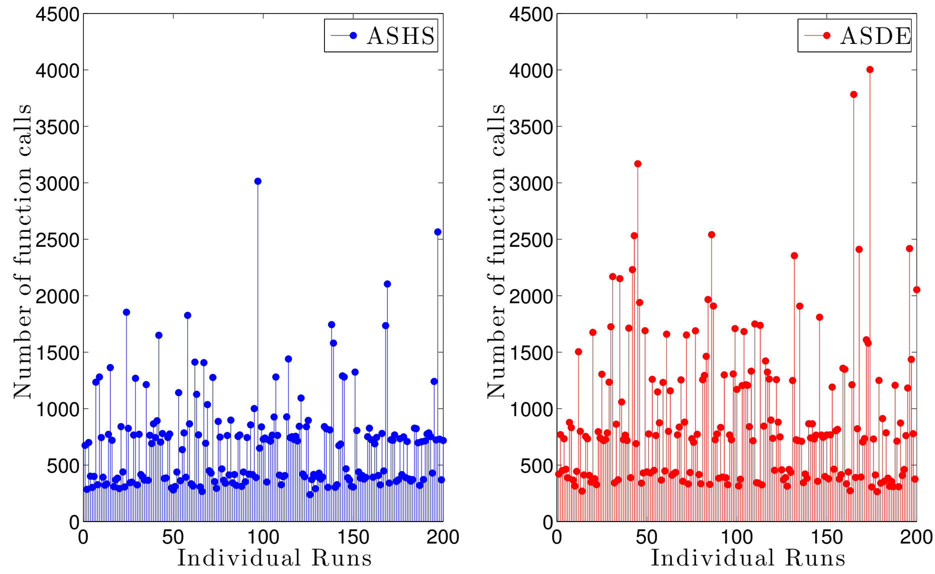

FIGURE 5.2: The number of function calls needed to reach $\epsilon_{tol} = 0.15$ for 200 runs of ASDE (red) and ASHS (blue).

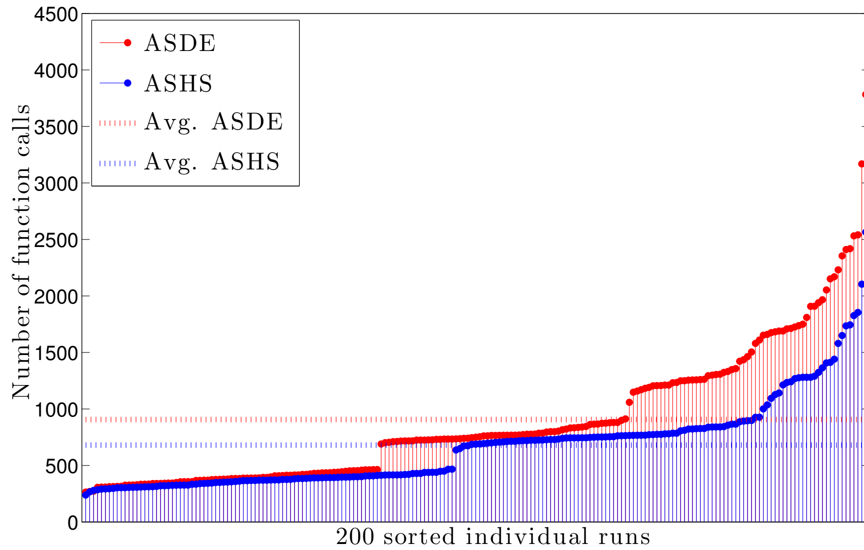

FIGURE 5.3: The number of function calls needed to reach $\epsilon_{tol} = 0.15$ for 200 runs of ASDE (red) and ASHS (blue). The values have been sorted in terms of number of function calls to increase readability.

ASDE and ASHS. Returning to Figure 5.3, we can analyse what made the average speed of ASDE lower than of ASHS.

Firstly, it can be seen that the ASHS manages to meet the stop criteria during the

first use of DS far more often than ASDE. The same thing is true for the second use of DS. To illustrate the nature of the hybrids, a minimization problem of one variable will be used. The objective function and stop criteria of the problem is shown in Figure 5.4.

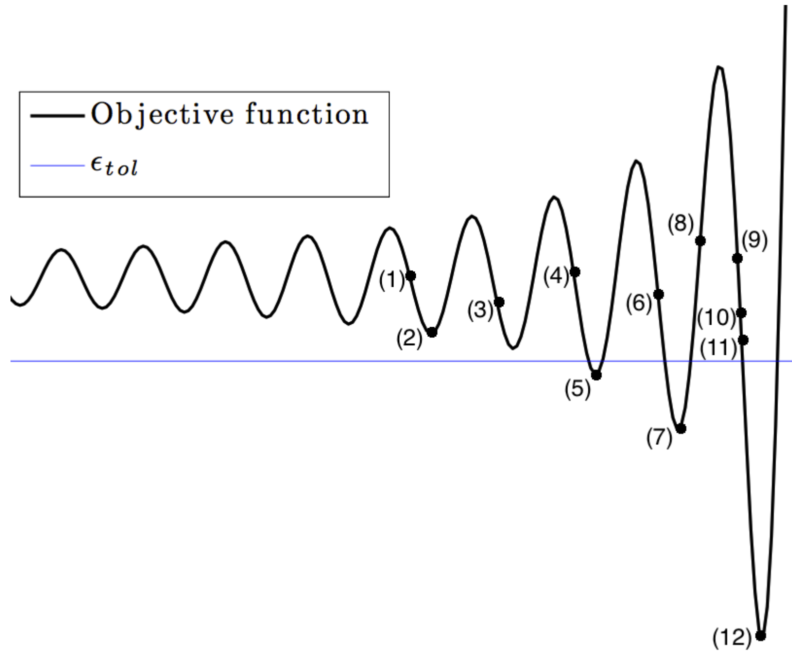

FIGURE 5.4: Illustration of a minimization problem with $D = 1$. The black line is the objective function, and the blue line is the stop criteria, ϵ_{tol}.

The global minimum is found at (12). We can see that both (5) and (7) are below ϵ_{tol} even though they don't represent the global minimum. (2) is an example of a local minimum that is not lower than ϵ_{tol}, and thus not an acceptable solution. Let's imagine that the best solution after the global part of one of the hybrids is (1). (1) is thus used as starting point in DS. Since DS is a local optimizer, no matter how many function calls it has in disposal, it will never manage to find a solution below ϵ_{tol}. If the starting point is (9) or (10) however, it will only be a question of time before DS crosses ϵ_{tol}. If the number of function calls needed to cross ϵ_{tol} with (9) as a starting point exceeds N_{local}, the hybrid will not reach ϵ_{tol} during the first use of DS. If (10) on the other hand is close enough to ϵ_{tol} to be reached by DS, this would greatly decrease the number of function calls needed to reach the stop criterion. Since ASDE and ASHS uses the exact same DS, the deviation in performance must come from the quality of DS's starting point. Based on Figure 5.4 we can separate the quality of a starting point into two main categories: quality in terms of exploration and quality in terms of exploitation. Starting points corresponding to (1) and (3) have poor quality in terms of exploration since DS

won't be able to cross ϵ_{tol} no matter how large the value of N_{local} is. IStarting points corresponding to 4 6 and 11 however are high quality points in terms of exploration. If we compare the starting points corresponding to (9) and (10), the quality in terms of exploitation is higher in (10) than in (9). Even though the geoacoustic inversion problem presented in this thesis have $D = 11$, the same ideas as in this simple example apply. This gives two theories to why ϵ_{tol} is reached during first and second use of DS more often in ASHS than in ASDE:

(A) : The starting points given by ASDE is of lower quality than the starting points of ASHS in terms of exploitation. This means that the starting points given by DE more often than the starting points of HS are further away from ϵ_{tol}. This gives the DS of ASHS a head start which makes ASHS cross ϵ_{tol} during the first use of DS more often than ASDE.

(B) : The starting points given by ASDE is of lower quality than the starting points of ASHS in terms of exploration. This means that the starting points given by DE more often than the starting points of HS are in a valley where the local minimum is larger than ϵ_{tol}.

If (A) was true, increasing N_{local} would lead to improved results for ASDE. To investigate this ASDE was run another 200 times, this time with $N_{local} = 380$. The average number of function calls needed was then reduced to 827. Since the increment of N_{local} led to better results, it is reasonable to think that some runs in the previous test with $N_{local} = 254$, did not meet the stop criteria during the first use of DS because it didn't have enough disposable function calls to complete the descent. Figure 5.5 shows a comparison of ASDE with $N_{local} = 254$ and $N_{local} = 380$.

The figure clearly shows that increasing N_{local} leads to more runs reaching the stop criteria during the first use of DS. This implies that the average starting point given from HS has higher quality than the average starting point given from ASDE in terms of exploitation. ASHS was also tested with $N_{local} = 380$ to see if increased N_{local} had any effect on performance. The average number of function calls over 200 runs was reduced to 640, which makes it clear that N_{local} should have been higher for both of the hybrids. This does however not change the fact that ϵ_{tol} was reached during the first use of DS by ASHS more often than ASDE.

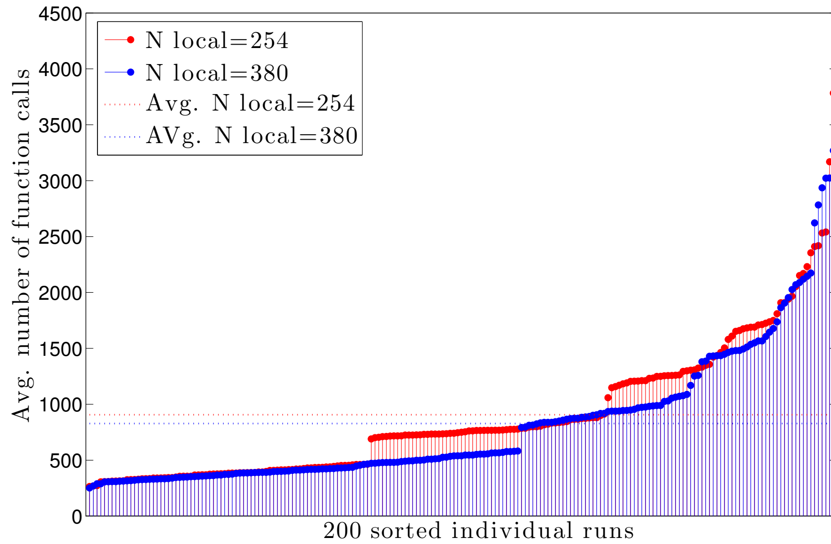

FIGURE 5.5: The number of function calls needed to reach $\epsilon_{tol} = 0.15$ for 200 runs of ASDE with $N_{local=254}$ (red) and ASDE with $N_{local} = 380$ (blue). The values have been sorted in terms of number of function calls to increase readability.

From this we can conclude that at least some part of ASHS's superior performance is in terms of exploitation and that (A) thus is true.

Figure 5.3 also shows that the worst runs of ASDE are worse then the worst runs of ASHS. This obviously lowers the average performance of ASDE relative to ASHS. The fact that ϵ_{tol} is not reached after multiple uses of DS can have two reasons. First lets imagine that (9) in Figure 5.4 is the result of DS after N_{local} function calls. DS was then stopped when it was descending towards the global minimum. The global optimizer would then use this solution, along with the other solutions stored in its memory (population for DE and harmony memory for HS), to further search the solution space. If the global optimizer then finds (3) or (1) after N_{global} iterations, this would lead the search away from a valley containing objective values lower than ϵ_{tol}. This problem would be solved by increasing N_{local}. From Figure 5.5 it can be seen that increasing N_{local} hardly had any effect on the frequency of bad runs. Therefore we can conclude that being sent away to a valley that don't include objective values lower than ϵ_{tol} is not a problem. Secondly, the bad performing runs might indicate that the hybrids in some runs are stuck in local optima. In Figure 5.4 this could mean that ASDE has found (2). In order to get out of the local minimum, the hybrids would have to find a point of lower ϵ,

e.g. (11). This will demand skills in terms of exploration. Since increasing N_{local} didn't decrease the frequency of bad performing runs, it is reasonable to believe that HS is superior to DS in terms of exploration. This would mean that also (B) is true.

The performance of metaheuristics are problem dependent; one algorithm might be good at some problems, while another is good at other problems. Because of this it is difficult to point out specific parts of the algorithms that makes one better than the other. Even though the performance of ASHS was consistantly higher than of ASDE when solving the geoacoustic problem, ASDE might outperform ASHS in other problems. This was in fact the case for the Rosenbrock problem of two variables.

CHAPTER 6

CONCLUSION

Hybrid versions of both harmony search (HS) and artificial bee colony (ABC) have been adopted to the problem of estimating geoacoustic parameters. The results show that the performance of the hybrids is much better than the performance of their respective global optimizers.

The tests further show that ASHS is superior to ASDE and ASABC both in terms of convergence speed and accuracy, when solving the geoacoustic problem. ASABC had the worst performance of the three hybrids. The tests indicates that both the exploiting and exploring abilities of ASHS's global optimizer, HS, are superior to the exploring and exploiting abilities of ASDE's global optimizer, DE. It should however be emphesized that the results from this thesis only show that ASHS does a better than ASDE at solving this spesific problem. When solving a benchmark problem the performance of ASDE was in fact better than the performanc of ASHS. This can be explained with the fact that the complexity of the benchmark problem was very low compared to the complexity of the geoacoustic problem.

Substantial tests on control parameter settings have been made in order to find optimal control parameter combinations. These tests have shown that most of the control parameters are highly sensitive. In most cases the settings found through these setting-tests gave good results later in the real algorithm testing. Some settings did however prove to be non-optimal. Future work should thus include further control parameter testing where more than five runs for each setting is used to calculate the average performance.

Control parameter tests suggests that an adaptive N_{local} had little effect on performance. This is probably due to the fact that most optimization runs reached ϵ_{tol} during their first and second use of DS. The adaptive N_{local} might thus have a bigger impact on problems where more hybrid iterations are made.

Since the tests performed in this thesis were performed with the use of synthetic reflection loss data, the next step for comparing ASHS with ASDE should include tests on real reflection loss data. Future work can also include a perturbation step inside of DS, to see if this further increases the performance of DS.

Future work should include the use of the bat algorithm (BA) [26], which is similar to HS, but more complex.

BIBLIOGRAPHY

[1] Barbosa H. J. C. Lavor C. C. and Raupp F. M. P., 2005, A GA-Simplex Hybrid Algorithm for Global Minimization of Molecular Potential Energy Functions, Annals of Operations Research, September 2005, Volume 138, Issue 1, pp 189-202, ISBN: 0-19-513159-2

[2] Bonabeau M., Dorigo M. and Theraulaz G., 1999, Swarm Intelligence: From Natural to Artificial Systems, Oxford University Press, ISBN: 0-19-513159-2

[3] Brekhovskikh L. M, 1980, Waves in Layered Media, 2nd ed., Academic New York

[4] Dong H., Ross Chapman N., Hannay David E., Rosso and Stan E., 2010, Estimation of Seismic Velocities of Upper Ocenic Crust from Ocean Bottom Reflection Loss Data, J. Acoust. Soc. Am. Volume 127, Issue 4, pp. 2182-2192

[5] Dorigo M. and Maria G., 1997, Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem, IEEE Trans. Evol. Comput., 1:5366,

[6] Dosso S. E., Wilmut M. J., Lapinski A. S., 2001, An Adaptive-Hybrid Algorithm for Geoacoustic Inversion, IEEE Journal of Oceanic Engineering, Vol. 26, No. 3, July 2001

[7] Geem Z. W, 2009, Music-inspired Harmony Search Algorithm, Springer, ISBN: 978-3-642-00185-7

[8] Geem Z. W, Harmony Search Source Code, Downloaded: 08.02.2013, https://sites.google.com/a/hydroteq.com/www/HS_Code_Matlab.zip?attredirects=0

[9] Haupt R. L. and Haupt S. E., 2004, Practical Genetic Algorithms, Second Edition, John Whiley & Sons Inc, ISBN 0-471-45565-2

[10] Holland J. H., 1992, Adaptation in Natural and Artificial Systems, Bradford Books, ISBN: 0-262-58111-6

[11] Jiang Y. Chapman N. R. and Gerstoft P., 2010, Estimation of Geoacoustic Properties of Marine Sediment Using a Hybrid Differential Evolution Inversion Method, IEEE Journal of Oceanic Engineering, Vol. 35, No. 1, January 2010

[12] Kang F. Li J. Xu Q., 2009, Structural Inverse Analysis by Hybrid Simplex Artificial Bee Colony Algorithms, Computers & Structures Volume 87 Issues 1314 July 2009 Pages 861870

[13] Karaboga D., 2005, An Idea Based on Honey Bee Swarm for Numerical Optimization, Erciyes University Engineering Faculty Computer Engineering Department, TECHNICAL REPORT-TR06 OCTOBER

[14] Karaboga D. and Akay B, 2012, A Modified Artificial Bee Colony Algorithm for Real-Parameter Optimization, Information Sciences Volume 192

1 June 2012 Pages 120142

[15] Karaboga D, Artificial Bee Colony Algorithm Source Code, Downloaded: 06.02.2013, http://mf.erciyes.edu.tr/abc/form.aspx

[16] Kirkpatrick S. Gelatt C.D. and Vecchi M. P., 1983, Optimization by Simulated Annealing, Science 220:671680

[17] Musil M. Wilmut M.J. and Chapman N. R., 1999, A Hybrid Simplex Genetic Algorithm for Estimating Geoacoustic Parameters Using Matched-Field Inversion, IEEE Journal of Oceanic Engineering, Vol. 24, No. 3, July 1999

[18] National Taiwan University Department of Computer Science Multimedia Information Retrieval LAB, 08.05.2013, http://neural.cs.nthu.edu.tw/jang/courses/cs4601/simplex.htm

[19] Nocedal J. and Wright S., 2006, Numerical Optimization, Second Edition, Springer, ISBN-13:978-0387-30303-1

[20] Parsopoulos K. E. and Vrahatis M. N., 2002, Recent approaches to global optimization problems through particle swarm optimization, Natural Computing, 1: 235306

[21] Storn T. and Price M. N., 1997, Differential Evolution - A simple and Efficient Heuristic for Global Optimization over Continous Spaces, Journal of Global Optimization 11: 341-359

[22] Storn T. Price M. N. Neimaier A. and Van Zandt J., Differential Evolution Source Code, Downloaded: 04.04.2013, http://mf.erciyes.edu.tr/abc/form.aspx

[23] Talbi E-G., 2009, Metaheuristics From Design to Implementation, John Wiley & Sons Inc., ISBN 978-0-470-27858-1

[24] Weise T., 2009, Global Optimization Algorithms Theory and Application, Second Edition, http://www.it-weise.de/

[25] Wu L. Wang Y. Yuan X. and Zhou S., 2010, A Hybrid Simplex Differential Evolution Algorithm, 2010 Chinese Control and Decision Conference

[26] Yang X.-S, 2010, A New Metaheuristic Bat-Inspired Algorithm, Studies in Computational Intelligence Springer Berlin 284 Springer 65-74 (2010)

[26] Yang X.-S, 2009, Harmony Search as a Metaheuristic Algorithm, Studies in Computational Intelligence, Springer Berlin, vol. 191, pp. 1-14 (2009)

Prepared in $\LaTeX\,2_\varepsilon$