Creative Design in Optimization

Metaheuristics Applied to Multi-modal Continuous Functions

Peter G. Schulz

Kongens Lyngby 2006

Abstract

The application of a number of metaheuristic methods to multi-modal continuous functions is described. Metaheuristics included are Descent Methods, Simulated Annealing, Tabu Search, Nelder–Mead simplex, Ant Colony Optimization, Genetic Algorithms, Evolution Strategies, Memetic Algorithms and Iterated Local Search. The Memetic Algorithm is chosen for implementation based on comparison and a modified SWOT analysis as it has the best possibilities of utilizing a divergent/convergent search strategy.

The problem solving process follows a creative approach where both brainstorming and SWOT analysis is used. A creative design process using brainstorming outlines the final algorithm. A creative search strategy based on divergent and convergent search is designed. In this way the metaheuristic search is guided around the search space. Two versions of the final algorithm are implemented. One, is the Memetic Algorithm with a divergent/convergent search strategy. Second, is a combination of the Memetic Algorithm and Simulated Annealing which is used to optimize parameter settings of the Memetic Algorithm.

The final algorithm is tested on five mathematically defined multi-modal continuous functions. The tests provide results similar to those found in the paper [14].

The final algorithm is used for parameter optimization of a groundwater simulation model. This case is used to verify the final algorithm on a black box problem, where the relation between input and output is hidden in the simulation model. Four different setups are used to compare manual calibration to that made by the metaheuristic approach. The objective function value has improved by 3 – 7 % in three out of four setups whereas, the objective value

has worsened by 5% for one setup.

It is seen that the final algorithm is useful for parameter optimization of groundwater models. Thus, it is concluded that the purpose of designing a method for optimization of multi-modal continuous functions is fulfilled. Furthermore, creative thinking has been utilized in the problem solving process and the search strategy design.

Finally, the project is evaluated in retrospective view. Parts of the design process is described taking departure in the creative thinking of professional designers studied by [5].

Resumé

Et antal metaheuristikkers anvendelse til optimering af kontinuerte funktioner med flere optima beskrives. Descent Methods, Simuleret Udglødning, Tabu-Søgning, Nelder–Mead simpleks, Myre Koloni, Genetiske Algoritmer, Evolutions-Strategi, Memetiske Algoritmer og Itereret Lokalsøgning er inkluderet i beskrivelsen. Den Memetiske Algoritme vælges til implementering baseret på sammenligninger og en modificeret SWOT analyse, idet den har de bedste muligheder for at udnytte en divergent/konvergent søgestrategi.

Problemløsningen følger en kreativ fremgangsmåde, der benytter sig af både brainstorming og SWOT analyse. En kreativ designprocess, der benytter brainstorming bruges til at frembringe den endelige algoritme. På den måde bliver metaheuristikkens søgning guidet rundt i løsningsrummet. Der implementeres to versioner af den endelige algoritme. Den ene er den Memetiske Algoritme med en divergent/convergent søgestrategi. Den anden er en kombination af den Memetiske Algoritme og Simuleret Udglødning, der bruges til at optimere parameter værdierne for den Memetiske Algoritme.

Den endelige algoritme testes på fem matematisk definerede kontinuerte funktioner med flere optima. Testene viser resultater svarende til dem der præsenteres i artiklen [14].

Algoritmen anvendes ydermere til parameter optimering af en grundvands simulerings model. Dette problem bruges til at verificere at algoritmen kan optimere et problem, hvor relationen mellem input og output er givet implicit i simulerings modellen. Fire forskellige opsætninger af problemet bruges til at sammenligne den manuelt kalibrerede model med modellen der er kalibreret vha. metaheuristikken. Værdien af objektfunktionen er forbedret med $3 - 7$ % i tre ud af fire

opsætninger hvorimod den er forværret med 5 % i et enkelt tilfælde.

Det fremgår at den endelige algoritme er nyttig i forbindelse med parameter optimering af grundvands simulerings modeller. Derfor konkluderes det at formålet med at designe en algoritme, der er i stand til at optimere kontinuerte funktioner med flere optima er opfyldt. Desuden er kreativ tænkning udnyttet i processen til problemløsning samt i det endelige design af søgestrategien.

I sidste ende evalueres projektet retrospektivt. Dele af designprocessen beskrives med udgangspunkt i den kreative tankegang for professionelle designere som er studeret af [5].

Preface

This master thesis was conducted at the Department of Informatics and Mathematical Modelling, the Technical University of Denmark under supervision of Reader René Victor Valqui Vidal, IMM. The project was carried out in the period from September 1^{st} 2005 to March 1^{st} 2006. The workload of the thesis is equivalent to 30 ECTS credits.

Acknowledgements

I would like to thank René Victor Valqui Vidal for advice and guidance during the project work and not at least for helping with finding literature relevant to the thesis. Furthermore, I would like to thank Associate Professor Philip Binning, Environment and Resources for sorting out the details in providing me with a license for GMS 5.1, and for being helpful in answering questions about the GMS software. I would like to thank Richard B. Winston, U.S. Geological Survey for answering questions about Modflow-2000 and referring to relevant documentation on Modflow-2000.

I would also like to thank my fiancée Minakshi Dhanda for providing the groundwater simulation model which, is used as a case for parameter optimization. Finally, I would like to thank Mikael O. Jensen, Minakshi Dhanda, and Tomas Netopil for proofreading.

Lyngby, March 2006

Peter G. Schulz

Contents

Chapter 1

Introduction

1.1 Purpose and Problem

This thesis seeks to combine creative thinking and methods of soft operations research with metaheuristic optimization. The aim is to optimize problems given by multi-modal continuous functions. The continuous function is defined by a continuous search space of variables in contrary to combinatorial optimization problems. This type of optimization problem can appear in different configurations. Pure mathematical functions can represent the multi-modal continuous properties. However, the continuous functions also appear in various real-life applications.

Simulation models of different kinds provide output which can be processed in an objective function. All simulation output is based on input parameters. Metaheuristic optimization of an objective function that rely on such simulations does not need to rely on a description or definition of the simulation model. The ability of obtaining an output from the simulation model for any input of parameters makes it possible to evaluate by an objective function. Problems where the relation between the input parameters and objective value is not described mathematically - or hidden by a simulation model - are referred to as black box problems in this thesis. The purpose of this thesis can be formulated by the following points:

- To design a method for optimization of multi-modal continuous functions. The method should be able to optimize both mathematically defined functions and black box functions where an explicit description of the relation between the solution input and the function output is not available.
- To merge creative thinking and methods of soft operations research with metaheuristic methods and mathematical optimization.
- Utilize creative thinking in the problem solving process and the implementation of the search strategy.

In this thesis a groundwater simulation model represents the real-life application. The choice of application is based on an idea developed through a case study made in a special course by Tomas Netopil and the author of this thesis. However, the simulation software used in this thesis is not the same as suggested in the special course.

1.2 Structure of the Report

The thesis is structured in chronological order to describe the problem solving process and the design of the metaheuristic method.

In chapter 2, it is described how different metaheuristics are applied to continuous functions. The description makes it clear what metaheuristics can be applied and how to do it. Furthermore, chapter 2 works as a foundation to the comparison and SWOT analysis presented in chapter 3. The SWOT analysis concludes by choosing which metaheuristic method to use.

The actual search strategy is developed in chapter 4. The design of the search strategy is initialized by a brainstorming session in order to come up with as many alternatives as possible and maintain fluency in the ideas. Four main tools are used for guiding the final algorithm - given in chapter 5 - in a divergent/convergent way.

In chapter 6, the results are presented. The final design of the algorithm is tested on five mathematically defined functions and a groundwater simulation model. The groundwater simulation model represents a black box function. The two types of results are compared to those of another metaheuristic presented by [17] and to those found by manual calibration respectively.

Chapter 7 outlines different discussion points and a conclusion on the work is given in chapter 8. The design process is evaluated in chapter 9.

Chapter 2

Overview of Metaheuristics

2.1 Introduction to the Description of Metaheuristics

In the following a number of metaheuristics will be described. The aim is to give an introduction to the metaphors from nature and/or social behaviour that inspire the metaheuristics. Furthermore the procedure of the metaheuristic is described often supported by pseudocode. Minimization problems are assumed when nothing else is specified. This means the smaller the objective value of a solution is the better. Assuming that fitness is a measure of something "positive" the fitness of a solution $\mathbf{x}$ is given by: fitness $= -f(\mathbf{x})$, where $f(\mathbf{x})$ is the objective value. If the fitness value is negative it can in some cases be necessary to shift the values by adding a constant to make them positive.

The description of the procedure is focused on how to make the metaheuristic work on continuous functions with a large number of local minima. Thus, the description differs from the one given in books on combinatorial optimization. The reason is that this section is seen as a prerequisite for choosing a metaheuristic for optimization of a continuous function. Naturally the pros and cons of using a metaheuristic on combinatorial problems, can be very different from the pros and cons of applying the same metaheuristic to continuous functions with multiple minima.

2.2 Neighborhood Based Metaheuristics

The first category of metaheuristics described in this thesis is based on neighborhood search or point-to-point movement. A characteristic of these metaheuristics is that the candidate solutions are somehow created by going from one solution to another related solution. The relation between a current solution and a candidate solution are diverse. Furthermore, the way to accept or reject candidate solutions are different. In fact the rejection/acceptation procedure is often closely related to the actual algorithm and its name.

The neighborhood based metaheuristics described in the following are all well known in combinatorial optimization. The simplest is the *Descent Method* (DM) or *Hill Climber* (HC). More interesting are the *Simulated Annealing* (SA) and the *Tabu Search* (TS) which can escape local minima.

2.3 Descent Methods

2.3.1 Metaphor

Descent Methods[1], Local Search (LS)[2] or Hill Climbing (HC) as it is called for maximization problems has its analogy to nature in the hill climbing metaphor. The hill climber in its simplest form is an individual who wants to reach the top. This is done by all the time taking a step in some random direction if that step brings the individual higher. This is referred to as a *greedy* version of the hill climber as it takes any step that is better not considering other possible steps. Another version of the hill climber could have a look at all possible steps to make and take the one that brings the individual highest referred to as the *steepest* version.

2.3.2 Descent Methods for Optimization of Continuous Functions

In order for a Descent Method to work on continuous problems a step has to be defined. If $\mathbf{x}^{(h)}$ is a vector representing a solution to the optimization

[1] minimization
[2] both minimization and maximization

problem at some iteration h the next improved solutions is represented by $\mathbf{x}^{(h+1)}$. Pseudocode 2.3.1 illustrates how the Descent Method works.

A way to find the next improved solution is to take a random candidate $\mathbf{x}^*$ from a neighbourhood defined by $\mathbf{x}^{(h)} + d \cdot \mathbf{z}$ where $\mathbf{z}$ is a symmetric vector of random numbers with mean zero, pseudocode 2.3.1 line 4. The factor d is determining the step size and can be reduced for every j'th iteration throughout the running of the algorithm if intensification is desirable, line 3 and line 11. In the case where the objective value of the candidate solution $f(\mathbf{x}^*)$ is lower than the the objective value of the current solution $f(\mathbf{x}^{(h)})$, the candidate solution is chosen as the new solution $\mathbf{x}^{(h+1)}$, line 6. The Descent Method stops when a local minimum is reached. Working with continuous functions this means that the objective value has not improved over a number of iterations N. Furthermore for methods utilizing intensification the step size d has to be below a specified threshold ϵ in order for the procedure to terminate, line 2.

Pseudocode 2.3.1 Procedure descent()

```
1:  x^(0) = random solution
2:  while f(x^(h)) < f(x^(h−N)) or d > ε do
3:     for h = h to h = h + j do
4:        x* = x^(h) + d · z
5:        if f(x*) < f(x^(h)) then
6:           x^(h+1) = x*
7:        else
8:           x^(h+1) = x^(h)
9:        end if
10:    end for
11:    d = βd, 0 < β < 1
12: end while
13: return x^(h)
```

2.3.3 Variations of Descent Methods

The Descent Method is probably the simplest metaheuristic. For combinatorial problems the solutions could be represented by a binary vector. A neighborhood is then defined by the solutions that are reached by flipping a single bit in the binary vector. For that kind of neighborhood intensification is not defined by a step size[3]. Thus, the algorithm becomes even simpler. The local minima

[3]intensification can be done in other ways i.e. by going from a *2-optimal* neighborhood, containing all the solutions occurring by flipping any combination of two bits, to 1-optimal neighborhood where a single bit is flipped

is easily defined for combinatorial problems by the iteration where no better solution is found in the entire neighborhood. This feature leads to the two different versions implied in section 2.3.1. The greedy version accepts a new solution $\mathbf{x}^*$ as soon as it is better than the current best solution $\mathbf{x}^h$. This is along the lines of the definition of the Descent Method in section 2.3.2. The steepest descent searches the entire neighborhood and accepts the solution $\mathbf{x}^*$ which improves the objective the most, given that such a solution exists. This is easily done for a neighborhood defined by flipping exactly one of the bits in a binary vector representing the solution. There is a finite (and small) number of solutions in the neighborhood.

The definition of the neighborhood on continuous variables in section 2.3.2, is inspired by the one used for Simulated Annealing by [17]. In this case there is an infinite number of potential candidates. Thus, the whole neighborhood cannot be searched. A compromise could be the combination of the steepest and the greedy version, where a limited number of randomly chosen candidates from the neighborhood are competing to be accepted as the best one. This approach is suggested amongst others for a combinatorial problem by [2].

2.3.4 Summary of the Pros and Cons of Descent Methods

The main reason for using Descent Methods is the simplicity. The neighborhood definition for continuous functions has some drawbacks when a steepest descent is desirable. However, a far more serious drawback is the fact that the Descent Method does not have the ability to escape local minima. If a local minimum is reached the algorithm terminates by definition. On the other hand a Descent Method can be useful for many problems where global optima is not necessary or if few local minima exist. A random restart could to some extent improve the issue of escaping local minima by given the procedure a number of chances to find different local minima. The best local minimum would then be returned after a number of random restarts.

2.4 Simulated Annealing

2.4.1 Metaphor

Simulated Annealing (SA) has its analogy to physics. Simulated Annealing was first used to simulate the annealing procedure of steal or other metals as they are cooled down in a heat bath. When heated to above the melting point the atoms

of steal are disordered and moves out of the grid structure for solid metals. When cooled down the atoms are again placed in an ordered structure. However, the structure depend on the rate of cooling. The slower the lower the final energy state of the structure is i.e. fast cooling will lead to a number of imperfections in the structure of the steal[6]. This means the process actually get trapped at a local energy minimum for fast decreases in temperature, whereas the global minimum state is likely to be reached when the cooling is slow. The laws of thermodynamics determines the probability of moving to a higher energy state throughout the process. The probability is given by equation (2.1) [6], where k is the Boltzmann constant, T is the temperature and ΔE is the difference in energy between the two states.

$$p(\Delta E) = \exp\left(\frac{-\Delta E}{k \cdot T}\right) \tag{2.1}$$

This feature can be used in optimization where the usual goal is to reach the global optimum instead of getting trapped at the first local optimum reached. At least having the possibility to reach better local optima is crucial.

2.4.2 Simulated Annealing for Optimization of Continuous Functions

Simulated Annealing can be seen as an extension of the descent methods. Whereas the descent methods only accept an improved solution, i.e. a solution with a lower objective value, Simulated Annealing can also accept and move to a solution with a higher objective value with a certain probability. The probability of accepting a given worse solution $\mathbf{x}^*$ than the current solution $\mathbf{x}^{(h)}$ at iteration h is described by an interpretation of equation (2.1) where $\Delta E = f(\mathbf{x}^*) - f(\mathbf{x}^{(h)})$, see equation (2.2)

$$p\Big(f(\mathbf{x}^{(h+1)}) = f(\mathbf{x}^*)\Big) = \exp\left(-\frac{f(\mathbf{x}^*) - f(\mathbf{x}^h)}{k \cdot T}\right), \quad f(\mathbf{x}^*) > f(\mathbf{x}^{(h)}) \tag{2.2}$$

As for the annealing process of metals, the results of Simulated Annealing depend on the cooling rate.

For continuous functions a possible layout of Simulated Annealing procedure is given by pseudocode 2.4.1 The parameters $\alpha, \beta, \epsilon, d$ and T are assumed to be initialized at the start of the procedure. As for descent methods a new candidate solution is generated by letting $\mathbf{x}^* = \mathbf{x}^{(h)} + d \cdot \mathbf{z}$ where $\mathbf{z}$ is a symmetric vector of uniformly distributed random numbers with mean zero, pseudocode 2.4.1 line 4. If the candidate solution $\mathbf{x}^*$ has a lower objective value than the overall best

solution $\mathbf{x}^{(h,\text{best})}$; the overall best solution is updated for the following iteration, line 5–9. In the case where the candidate solution has a lower objective value than the current solution, a descent move is made in order to reach the candidate solution, line 10–11. On the other hand if the objective of the candidate is larger than the one of the current solution; an ascent move is accepted with the probability described by line 14, where the Boltzmann constant is replaced by a normalizing constant c. Note the exponential expression is also simplified in relation to equation (2.2). Thermal equilibrium is reached by the while loop in line 3, since the loop only continues as long as the overall best solution improves at least every N moves. Note that the while loop is considered to hold for all h smaller than N since the solution $\mathbf{x}^{(h-N,best)}$ will not be defined in that case. In the outer while loop the temperature and step size are decreased, line 24–25. This process continues as long as the temperature is above a certain threshold ϵ, line 2. Finally, the overall best solution is returned, line 27.

Pseudocode 2.4.1 Procedure SA()

```
1:  x^(0) = x^(0,best) = random solution
2:  while T > ε do
3:     while f(x^(h,best)) < f(x^(h−N,best)) do
4:        x* = x^(h) + d · z
5:        if f(x*) < f(x^(h,best)) then
6:           x^(h+1,best) = x*
7:        else
8:           x^(h+1,best) = x^(h,best)
9:        end if
10:       if f(x*) < f(x^(h)) then
11:          x^(h+1) = x*
12:       else
13:          draw random χ from uniform interval [0; 1]
14:          if χ < (1/c) exp((f(x^(h)) − f(x*))/T) then
15:             x^(h+1) = x*
16:          else
17:             x^(h+1) = x^(h)
18:          end if
19:       end if
20:       h = h + 1
21:    end while
22:    x^(1) = x^(h,best)
23:    h = 1
24:    T = αT, 0 < α < 1
25:    d = βd, 0 < β < 1
26: end while
27: return x^(h,best)
```

As the temperature decreases over each iteration of the outer loop the right hand side of the expression in line 14 pseudocode 2.4.1 approaches zero. Thus, the probability of accepting ascent moves decreases. At high temperatures a local minimum is easily escaped and the algorithm moves around large parts of the search space whereas a local minimum for low temperatures is only rarely escaped. The expectation is that for low temperatures the search is concentrated close to the global minimum. Thus, the property of intensification in a local area is desirable at that point[4].

2.4.3 Variations of Simulated Annealing

Simulated Annealing for continuous functions described in section 2.4.2 allows an ascent move with a certain probability although a descent move might be possible in the neighborhood of the current solution. The reason is that the continuous nature makes it impossible to compare all solutions in a neighborhood as it is done for combinatorial optimization problems. A way to get around this drawback is described by [17] who combines Simulated Annealing with a *Nelder-Mead* simplex algorithm.

Several versions of Simulated Annealing exist. The main differences are in the definitions of the acceptance functions and neighborhoods. A basic approach where the normalizing constant is omitted is referred to by [6]. An approach called adaptive Simulated Annealing (ASA) is described by [16] who has developed the fast annealing (FA) further.

2.4.4 Summary of the Pros and Cons of Simulated Annealing

Simulated Annealing is another easy-to-implement metaheuristic. Basically it is a descent method added a probabilistic acceptance function. This makes Simulated Annealing able to escape local minima. Because of this ability combined with the intensification provided by cooling Simulated Annealing is likely to reach better local optima than by using a descent method from a random starting point. In fact Simulated Annealing have the advantage over other methods by its relation to thermodynamic theory. In theory a cooling *schedule* can be made to guarantee asymptotic convergence [6]. However, the convergence turns out to require exponential solution times according to problem size in practice.

[4]The relation between convergent/divergent thinking and diversification/intensification is described in section 4.3

2.5 Nelder–Mead Simplex

2.5.1 Metaphor

The Nelder–Mead simplex should not be mistaken for the well known simplex algorithm developed for linear programming by Dantzig. The Nelder–Mead is a direct search method. Thus, it can be viewed as another interpretation of the Hill Climber metaphor.

2.5.2 Nelder–Mead for Optimization of Continuous Functions

In the Nelder–Mead procedure the *simplex* consists of exactly $n+1$ solution vectors $\mathbf{x}_k, k = 0, \ldots, n$, where n is the number of decision variables or length of the solution vector. Letting each of the $n+1$ solutions represent a point in the search space will form a geometrical object in n dimensions called the simplex. Each operation of the Nelder–Mead simplex procedure is performed on an entire solution vector. As an analogy to the evolutionary algorithms the simplex in the Nelder–Mead procedure can be viewed as a population of individuals. However, the number of individuals are $n+1$ in this case.

The Nelder–Mead simplex procedure is given by pseudocode 2.5.1. The procedure takes an initial simplex as argument and returns the best solution in a another final simplex, line 34 pseudocode 2.5.1. The parameters ρ, χ, γ and σ are assumed initialized at the start of the procedure. The parameters should satisfy:

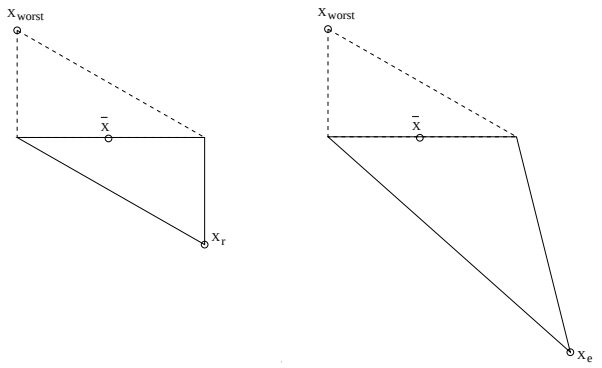
$$\rho > 0, \chi > 1, \chi > \rho, 0 < \gamma < 1, \quad \text{and} \quad 0 < \sigma < 1$$

In fact a common universal choice of parameters are according to [19] :

$$\rho = 1, \chi = 2, \gamma = \tfrac{1}{2} \quad \text{and} \quad \sigma = \tfrac{1}{2}$$

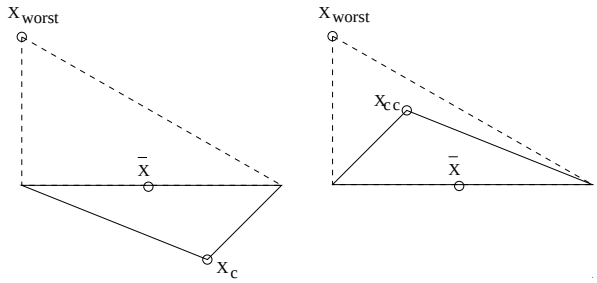
The Nelder-Mead procedure runs until a stopping criterion is satisfied, line 1. The stopping criterion is usually connected to a measure of how far the simplex has moved from one iteration to the following. Each iteration starts by ordering the solutions in the simplex increasingly by the objective values, line 2. Secondly the *centroid* $\bar{\mathbf{x}}$ is calculated based on the average of the n best solution points, i.e. all points except the worst $\mathbf{x}_{n+1}$, line 3. In order to accept a new solution

into the simplex a *reflection point* $\mathbf{x}_r$ of the worst point is calculated, line 4. The reflection point is illustrated by figure 2.1(a). If the objective value of the reflection point is in between the objective value of the best point and the objective value of the second to worst point the reflected point is accepted right away, line 5–6. In case the objective value of the reflection point is strictly less than the objective of the best point in the simplex a new *expansion point* $\mathbf{x}_e$ is calculated, line 8–9. The expansion point is illustrated by figure 2.1(b). The better of the reflection and the expansion point is accepted to the simplex on expense of the worst point $\mathbf{x}_{n+1}$, line 10–14. If the objective value of the reflection point is larger than the second to worst point $\mathbf{x}_n$ in the simplex a contraction is performed, line 15. In case the reflection point has an objective that is strictly smaller than the objective of the worst point a *contraction point* $\mathbf{x}_c$ situated outside the simplex is calculated, line 16–17. The outside contraction point is illustrated by figure 2.2(a). The contraction point is only accepted on expense of the worst point if it has an objective smaller than the reflection point, line 18–19. Otherwise the simplex is shrinked, line 21. The shrink sub-procedure is given by pseudocode 2.5.2 in section 2.5.2.1. In case the reflection point has an objective at least as large as the worst point the contraction point $\mathbf{x}_{cc}$ placed inside the simplex is calculated, line 23–24. The inside contraction point given by figure 2.2(b) is only accepted if the objective is strictly smaller than the worst objective, line 25–26. Otherwise the simplex is shrinked, line 28. The Nelder–Mead procedure returns the solution point from the final simplex which has the smallest objective value, line 34.



(a) Reflection point (b) Expansion point

Figure 2.1: Reflection and expansion in the Nelder–Mead simplex



(a) Outside contraction point (b) Inside contraction point

Figure 2.2: Contraction in the Nelder–Mead simplex

2.5.2.1 The Shrink Procedure

The shrink procedure is also referred to as *multi-contraction*. A shrink procedure is given by pseudocode 2.5.2. Only the best point $\mathbf{x}_1$ is kept in a shrink step, line 1 pseudocode 2.5.2. The following n points are moved closer to the best point, line 2–4. Figure 2.3 illustrates the simplex before and after a shrink.

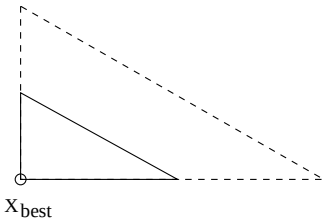


Figure 2.3: Shrinking simplex

A slightly simpler version of the Nelder–Mead simplex is described by [3] who utilize it in a combined tabu search approach. The method described by [3] does not differ between a reflection point better or worse than the second to worst point, i.e. only one type of contraction is performed.

Pseudocode 2.5.1 Procedure Nelder–Mead(Simplex)

```
1: while termination criterion not satisfied do
2:    Order Simplex = {x_1, ..., x_{n+1}} such that f(x_1) ≤ f(x_2) ≤, ..., ≤ f(x_{n+1})
3:    x̄ ← Σ_{k=1}^{n} (1/n) · x_k
4:    x_r ← x̄ + ρ(x̄ − x_{n+1})
5:    if f(x_1) ≤ f(x_r) < f(x_n) then
6:       Simplex ← {Simplex\{x_{n+1}}} ∪ {x_r}
7:    else
8:       if f(x_r) < f(x_1) then
9:          x_e ← x̄ + χ(x_r − x̄)
10:         if f(x_e) < f(x_r) then
11:            Simplex ← (Simplex\{x_{n+1}}) ∪ {x_e}
12:         else
13:            Simplex ← (Simplex\{x_{n+1}}) ∪ {x_r}
14:         end if
15:      else
16:         if f(x_r) < f(x_{n+1}) then
17:            x_c ← x̄ + γ(x_r − x̄)
18:            if f(x_c) ≤ f(x_r) then
19:               Simplex ← (Simplex\{x_{n+1}}) ∪ {x_c}
20:            else
21:               Simplex ← Shrink(Simplex,σ)
22:            end if
23:         else
24:            x_cc ← x̄ − γ(x̄ − x_{n+1})
25:            if f(x_cc) < f(x_{n+1}) then
26:               Simplex ← (Simplex\{x_{n+1}}) ∪ {x_cc}
27:            else
28:               Simplex ← Shrink(Simplex,σ)
29:            end if
30:         end if
31:      end if
32:   end if
33: end while
34: return({x_k | min f(x_k), x_k ∈ Simplex})
```

Pseudocode 2.5.2 Procedure Shrink(Simplex,σ)

```
1: NewSimplex ← {x_1}
2: for k = 2 to n + 1 do
3:    v_k ← x_1 + σ(x_i − x_1)
4:    NewSimplex ← NewSimplex ∪ {v_k}
5: end for
6: return(NewSimplex)
```

2.5.3 Pros and Cons of the Nelder–Mead Simplex Procedure

The Nelder–Mead simplex procedure is a direct search method as mentioned earlier. This means that the simplex will converge to a final non-improving simplex since each iteration in the Nelder–Mead procedure has to satisfy a descent condition. A drawback comparing to many metaheuristics is the inability to escape a local minimum. Furthermore the Nelder–Mead simplex is not proved to converge to a minimum for higher dimensions than one. In fact, according to [19] a family of strictly convex functions and starting conditions exist where the Nelder–Mead procedure is converging to a non-minimum in two dimensions. However, the Nelder–Mead simplex is very popular for optimizing continuous functions. This might be due to the ability of initially producing steeply descending moves in practice. Furthermore, each descending move does typically only require one or two function evaluations. Thus, the procedure is fast and can perform well. On the other hand the drawback that not even a local minimum can be guaranteed calls for hybridization with other metaheuristics which is often the case.

2.6 Tabu Search

2.6.1 Metaphor

A metaphor describing Tabu Search (TS) can be derived from the Hill Climber metaphor used for descent methods. Whereas the Hill Climber (maximization) would never make a step bringing the individual lower, i.e. away from the peak (local optimum) this is not the case for Tabu Search. The individual in Tabu Search can be seen as having a memory of already visited places – the tabu list. If a peak is already reached no steps can be made to bring the individual higher. In this case a downhill step can be made if it brings the individual to an unvisited place not on the list. After some time an already visited place is forgotten, i.e. deleted from the tabu list. Afterwards that place can be visited again.

The idea is to utilize the memory in order to leave a peak. The motivation to leave is to find an even higher peak, i.e. the property of escaping local optimum. The tabu list is used to control the way to leave a local optima whereas for Simulated Annealing this is done by the probability function determined by the energy function of annealing.

The word tabu means something that is banned or prohibited. A definition of tabu from Cassels Concise Dictionary is:

1. something which is very strongly disapproved of in a particular society etc.

Thus, the word tabu is related to a certain interpretation of something in society more than as the memory of an individual. However, having a list of tabus implies having a memory.

2.6.2 Tabu Search on Continuous Problems

As it seems simple to reject a solution to a combinatorial problem if it is included in the tabu list, this is not the case for continuous problems. As for other metaheuristics a random candidate solution within a neighborhood can be defined. If this solution has an objective value higher than the current solution (minimization), the decision whether to accept it or not is based on the content of the tabu list. However, rather than checking if the solution is already tabu it should be checked if the solution is within a certain distance of a solution in the tabu list. A TS algorithm with this property called *Enhanced Continuous Tabu Search* (ECTS) is given by [24].

2.6.2.1 The ECTS Procedure

An overview of the procedure of the ECTS is presented in pseudocode 2.6.1. As for the previous section $\mathbf{x}$ represents a solution point in the continuous solution space. The parameters occurring in the following are assumed to be initialized at the start of the procedure.

At first a number N_s of initial solutions $\mathbf{x}^{*i}$ are generated. This is done at random, line 2–3 pseudocode 2.6.1. The solution with the lowest objective value is at all times saved as $\mathbf{x}$, whereas the remaining solutions are inserted in the tabu list (TL), line 4–9. The final $\mathbf{x}$ of the initialization step is inserted in an empty promising list (PL), line 11.

After the initial solution generation the actual search runs until a stopping criterion is satisfied, line 12. This criterion would typically be the number of iterations or the amount of CPU-time spend. Each iteration starts by generating a number n of *neighboring* solutions to the current solution $\mathbf{x}$, line 13. A description of the GenerateNeighbors procedure is given in section 2.6.2.2. For each *neighbor* it is checked whether the neighbor solution N^i is within a dis-

tance r_{tabuball} of any solution $\mathbf{x}^{TL}$ in the tabu list, line 14–15. Furthermore it is checked whether the neighbor solution is within a distance r_{promball} of any solution $\mathbf{x}^{PL}$ in the promising list. The neighbor solution is rejected in either case, leaving only neighbor solutions that has at least a certain distance to already visited solutions, i.e. solutions in tabu and promising lists. The new solution is chosen from the remaining neighbors as the solution with the lowest objective value, although it is not necessarily lower than the one from the preceding iteration, line 19–27.

The last part of the procedure starts by identifying the solution $\mathbf{x}$ in the promising list that has the lowest objective value, line 30–35. The most promising of the promising solutions so to say. A local minimization procedure starts from this solution. As indicated by line 36 the local minimization procedure could be the Nelder–Mead simplex described in section 2.5 page 10.

2.6.2.2 Neighbor Generation in the ECTS

The n neighboring solutions $S^m, m \in \{1, \ldots, n\}$ can be generated randomly within n *hyper rectangles* each defined below as:

1. having the center in the current solution $\mathbf{x}$
2. having N dimensions, where N is the number of decision variables in the continuous optimization problem.
3. having the side length $2{\cdot}h_m$ defined by $h_{n-m+1} = \frac{h_n}{2^{m-1}}, \forall m \in \{1, 2, \ldots, n\}$, where h_n is a parameter initialized by the user.

An example of the hyper rectangles in the two dimensional case, i.e. the number of decision variables is two, is illustrated by figure 2.4. As indicated by figure 2.4 the number of neighboring solutions is three and only one solution lies within two adjacent rectangles. In order to obtain this property, the randomly generated solutions should be chosen with care. To view a detailed description of how to choose the randomly generated solutions see [24].

2.6.3 Variations of Tabu Search

Whereas the Tabu Search is widely used in combinatorial optimization only a few persons have applied it to optimization of continuous functions according to [18, 14]. However, several interpretations of the Tabu Search metaphor exists

Pseudocode 2.6.1 Procedure TS()

```
1: TL = ∅
2: for i = 1 to i = N_s do
3:    x^{*i} =random solution
4:    if f(x^{*i}) < f(x) then
5:       TL = TL ⋃ {x}
6:       x = x^{*i}
7:    else
8:       TL = TL ⋃ {x^{*i}}
9:    end if
10: end for
11: PL = {x}
12: while termination criterion not satisfied do
13:    S = GenerateNeighbors(x, n)
14:    for i = 1 to i = n do
15:       if x^{TL} ∈ TL exists | (r_tabuball > |S^i − x^{TL}|) or x^{PL} ∈ PL exists | (r_promball >
          |S^i − x^{PL}|) then
16:          S = S\{S^i}
17:       end if
18:    end for
19:    x = S^1
20:    for i = 2 to i = |S| do
21:       if f(S^i) < f(x) then
22:          TL = TL ⋃ {x}
23:          x = S^i
24:       else
25:          TL = TL ⋃ {S^i}
26:       end if
27:    end for
28:    PL = PL ⋃ {x}
29: end while
30: x = PL^1
31: for i = 2 to i = |PL| do
32:    if PL^i < x then
33:       x = PL^i
34:    end if
35: end for
36: NelderMead(x)
```

for continuous functions. A Directed Tabu Search (DTS) is presented by [14]. Comparing the ECTS by [24] the DTS utilizes even more lists, subroutines and strategies. The DTS generates trial points in the whole search space based on a *diversification* scheme whenever the Tabu Search does not produce any improvement otherwise . The diversification scheme roots in a *visited region list* storing previously visited regions of the search space. Another difference between the

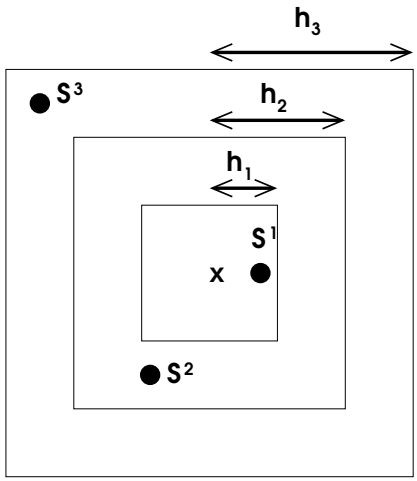


Figure 2.4: The Hyper rectangles in Two Dimensions, Where Three Neighboring Solutions are Created

two procedures is the ability of the DTS to direct the neighborhood search out of so called *semi tabu regions*, i.e. the regions surrounding the tabu regions defined by a ball with a certain radius r_{tabuball}. A crucial difference between the two is that DTS uses a local minimization technique like the Nelder–Mead or the *Adaptive Pattern Search* (APS) to conclude the neighborhood search at each iteration.

The similarities between the two are also clear. Both procedures define a tabu region for all dimensions around each point in the tabu list. Both make some kind of exploration of the neighborhood around the current solution. Furthermore the entire search procedure is finished by a local minimization technique like a version of the Nelder–Mead simplex, described in section 2.5 page 10 or a quasi-Newton approach.

A third approach to use Tabu Search for optimizing continuous problems is described by [18]. The approach called *Memory Tabu Search* (MTS) considers a move tabu if it is within a certain distance of the current solution or even if the objective value is not lowered more than a certain fixed value or a certain percentage. This means only improving moves are allowed in contrary to the two other procedures.

2.6.4 Summary of the Pros and Cons of Tabu Search

The Tabu Search is a fast metaheuristic for combinatorial problems because it is easily evaluated whether a solution is included in the tabu list or not. For continuous functions more calculations has to be made in order to produce a new solution. Examples of such calculations could be the distance and generation of search directions. This is of course time consuming on the negative side. Although, a lot less function evaluations are made comparing to population based approaches, as seen from [24], competitive results are reached. The major drawback comparing to any other non-memory based metaheuristic is the need of allocating memory for saving solutions that might represent a considerable amount of data. This is in some implementations avoided by saving the *move* from one solution to another, referred to as attribute based memory in [11]. However, this is complicated for continuous functions and is not seen in any of the referenced papers.

2.7 Swarm Intelligence

The second category of metaheuristics presented in this thesis is based on *Swarm Intelligence* (SI) or *Multi Agent Systems* (MAS). The characteristic of Swarm Intelligence is the use of decentralized individual *agents* interacting with each other [25]. The artificial agents in Swarm Intelligence optimization are created by analogy to social insects and animals from nature e.g. bees, wasps, ants, termites, birds and fishes.

2.8 Ant Colony

2.8.1 Metaphor

The Ant Colony (AC) metaheuristic belongs to the category of Multi Agent Systems or Swarm Intelligence. The relation to the metaphor is rather strong. The idea is to let several agents try out various solutions and intensify the search in the promising regions.

A characteristic of Swarm Intelligence in general and ant colony in particular is that the movement of one agent, in this case an ant, is highly dependent on the movement of the previous ants. In an ant colony the search for food is done by a

number of scout-like ants [25]. When succeeding in finding food these ants leave a trail of the chemical substance *pheromone* that can be tracked by other ants. The cooperation between the ants in the ant colony is divided into a *feedback mechanism* and an *updating mechanism*. The feedback mechanism means that ants are more likely to choose a path with a high concentration of pheromone than a path with lower concentration of pheromone. The updating mechanism means that the amount of pheromone on a particular path increases by each ant traversing. Furthermore, the decrease of pheromone due to evaporation over time causes a negative updating.

A single ant can be viewed as a *reactive agent* since it follows a trail until it comes upon an intersection. When having the possibility to choose from two or more paths the ant will choose the one with the highest concentration by a higher probability. However, in artificial systems the agents often behave slightly different. The agents in artificial Swarm Intelligence and ant colonies can be either reactive or intelligent. The *intelligent agent* is characterized by having a goal or objective of its own. An example of such a property is seen in the Ant Colony Optimization (ACO) of the the combinatorial travelling salesman problem (TSP). When an ant has to choose which city to visit next on the travelling salesman tour it could consider, not only the concentration of pheromone left on the paths to possible cities, but also the distance to each city. That means the probability of choosing a path is increased by one objective; to choose the path that has the highest concentration, and decreased by another objective; to make the greedy choice of the shortest path (edge) to the next city. Thus, a weighting of the two objectives has to be made.

2.8.2 Ant Colony Optimization of Continuous Functions

The Ant Colony Optimization is intuitively well suited for combinatorial problems, due to the fact that an ant has to choose between two or more existing paths. Each path represent a solution component, i.e. a city in the travelling salesman problem, an item in the knapsack problem (KP) etc. In fact it is difficult to choose over continuous variables because of the obvious problem of how to distribute the pheromone trails in the continuous search space. The authors in [8] go as far as claiming that Ant Colony Optimization is suited for discrete problems alone. However, methods of applying ant colony to optimization of continuous functions do exist. Perhaps the most obvious way to get around this is to use discretization.

In this thesis an approach suggested by [1] is presented. This version of Ant Colony Optimization is a hybrid method utilizing genetic operations, see section 2.10 page 25, rather than a strict ant colony. In this approach the ant colony

is only used for local optimization whereas a genetic algorithm is suggested to make a global search, i.e. the genetic algorithm determines the location of the *nest* and thereby the starting point of the Ant Colony Optimization. Other simpler methods like Descent Methods or other metaheuristics described in this thesis, could also be used to locate the nest. Thus, in the following is focused on the ant colony part of the approach to solve continuous functions. In pseudocode 2.8.1 the details of the local Ant Colony Optimization are presented. Each ant k represents a solution vector $\mathbf{x}_k^{(t)}$ at a given time t. The parameters mentioned in the following are assumed initialized at the start of the procedure.

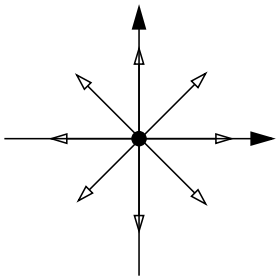


Figure 2.5: The initial search directions in a two dimensional nest neighborhood

The initial search directions represented by the vectors $\mathbf{d}_j^{(0)}$ are defined with equal length R^* in line 2. The search directions are as far apart as possible and cover all dimensions. An example of initial search directions in a two dimensional search space is given in figure 2.5. In this case the j'th search direction is defined by $\mathbf{d}_j = (\sin(j/m \cdot 2\pi), \cos(j/m \cdot 2\pi)) \cdot R^*$, where m is the total number of ants and search directions. In line 3–5 each ant k is sent in a search direction $\mathbf{d}_j^{(t)}$. For simplicity the number of ants m and search directions are assumed to be equal. If the number of ants were larger than the number of search directions, the ants should be sent in a search direction according to a probability function. As seen below this is the case for the following steps. The ants are sent out in search directions that are adapted at each step until a stopping criterion is satisfied. This criterion is the maximum number of time steps T and/or a minimum relative improvement of the best objective value, line 6. In line 8–10 the new pheromone trail is updated for each of the m directions. The updating is combined from an evaporation process and an adding process. The evaporation is defined by the relative *evaporation constant* ρ, where $(1 - \rho)$ is the amount of pheromone evaporated. The amount of new pheromone is determined by equation (2.3).

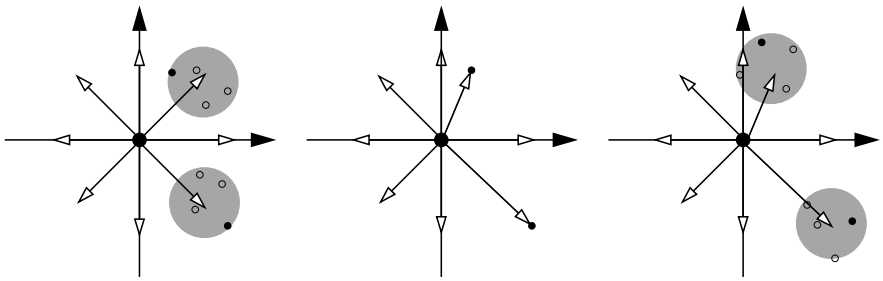
$$\Delta\tau_j^{(k,t)} = \begin{cases} \frac{1}{f(x_k^{(t)})} \cdot \text{const} & \text{if ant } k \text{ travels direction } j \\ 0 & \text{otherwise} \end{cases} \tag{2.3}$$

Meaning that if an ant k travels in the direction j the amount of pheromone laid by the ant k on the direction j is proportional to the fitness[5] of the solution represented by the ant. Each ant k (line 11) is sent into a search direction $\mathbf{d}_j^{(t)}$ by a probability given by equation (2.4), line 15–19.

$$P_{\mathbf{d}_j^{(t)}} = \frac{\tau_j^{(t)}}{\sum_{i=1}^{m} \tau_i^{(t)}} \tag{2.4}$$

The probability is defined by the relative amount of pheromone on each search direction. Thus, the larger amount of pheromone on a search direction the larger the probability of sending an ant in the direction. Note that more ants can be sent into the same search direction in this way. It should be emphasized that the ants do not stop after reaching the destination of the search direction. A random step $\Delta(t, R)$ is added, line 16–18. The random step lies within the range $[0; R]$, where R is the search radius. As seen from the definition in line 17 the probability of $\Delta(t, R)$ being close to zero increases by increasing t. This can be seen as a kind of intensification of the search over time. The parameter b determines the degree of non-uniformity of the random step.

The search directions are adapted at the end of each time step. The j'th search direction for the next time step $\mathbf{d}_j^{(t+1)}$ is set to point to the location of the best ant in the current time step that has been travelling the direction $\mathbf{d}_j^{(t)}$. Figure 2.6 illustrates the updating of two search directions in the two dimensional search space, where the search radius R is shown as a shaded circular region.



(a) Before adaption (b) Adaption (c) After adaption

Figure 2.6: Adaption of two search directions

[5] the fitness is in this case the reciprocal of the objective value

Pseudocode 2.8.1 Procedure ACO()

```
1: t ← 0
2: define initial search directions d_j^(t), ∀j ∈ {1,...,m} with equal length
3: for k = 1 to m do
4:    x_k^(t) ← nest + d_j^(t), ∀j = k
5: end for
6: while termination criterion not satisfied do
7:    t ← t + 1
8:    for j = 1 to m do
9:       τ_j^(t) ← ρ · τ_j^(t−1) + Σ_{k=1}^m Δτ_j^(k,t)
10:   end for
11:   for k = 1 to m do
12:      draw random χ from uniform interval [0; 1[
13:      j ← 1
14:      while x_k^(t) is not defined do
15:         if Σ_{i=1}^{j−1} τ_i^(t) ≤ χ · Σ_{i=1}^m τ_i^(t) < Σ_{i=1}^j τ_i^(t) then
16:            draw random r from uniform interval [0; 1]
17:            Δ(t, R) ← R(1 − r^{(1−t/T)^b})
18:            x_k^(t) ← nest + d_j^(t) + Δ(t, R)
19:         end if
20:         j ← j + 1
21:      end while
22:   end for
23:   for j = 1 to m do
24:      d_j^(t+1) ← v(nest, x_k^(t))|{min f(x_k^(t))∀x_k^(t) defined by d_j^(t) }
25:   end for
26: end while
27: return x_k^(t)|{min f(x_k^(t)), ∀k, t}
```

2.8.3 Variations of Ant Colony Optimization

The approaches to Ant Colony Optimization of continuous functions are very different. However, a similarity of many algorithms is the hybrid approach where ant colony is merged with another methodology in order to work for continuous problems. Where [1] used a genetic algorithm to locate the nest a more merged approach is presented by [4] who use *crossover* and *mutation* in order to develop the ants. A third hybridized approach by [10] merges the properties of ant colony and local search using the *Powell Method*. Discretization of the search space is used by [22] which makes the ant colony metaphor more directly interpretable.

The probability of the ants choosing a direction in the approach presented in this thesis rely only on the amount of pheromone. Thus, the ants are acting as reactive agents. It turns out that the agents of Ant Colony Optimization described by [4, 10, 1, 22, 28] are all reactive. However a continuous approach

suggested by [9] introduces heuristic distance information which is used by the ants. Taking the distance information into account the ants in [9] can be described as intelligent.

2.8.4 Pros and Cons of Ant Colony Optimization

The main drawback of Ant Colony Optimization of continuous functions is probably related to high consumption of resources due to the parallel calculations of many ants and poor convergence. The ant colony needs to search for good solution throughout the neighborhood of many paths – both promising and less promising because it can only be guaranteed that an ant will not use a path when all the pheromone has evaporated. This feature on the other hand ensures the diversity of the search i.e. the ant colony is not likely to get trapped in a local minimum.

Most ant colonies for continuous problems are poorly described. Often a great effort is made to explain the well known approach of ant colony for combinatorial problems. The extension to the continuous functions often lack information of the exact procedures. This could perhaps be seen as a sign that the ant colony is difficult to adapt to continuous functions.

For combinatorial problems the heuristic distance function is of great importance e.g. the travelling salesman problem where the probability of choosing a city is increased for nearby cities. The ability of utilizing intelligent agents are important for the results of the procedure. Keeping this in mind a drawback of the ant colony for continuous function optimization is the lack of intelligent agents.

2.9 Evolutionary Algorithms

In this section a group of metaheuristics called Evolutionary Algorithms (EA) is presented. The main characteristic of Evolutionary Algorithms is the evolvement of a population of individuals. In most cases the individuals create new offspring through crossover operations and mutation. Some categorise metaheuristics within Swarm Intelligence as Evolutionary Algorithms. However, in this thesis a distinction is made because, in Swarm Intelligence the following populations are not necessarily offspring of the current population. Furthermore the main characteristic of Swarm Intelligence is agents reaction to actions of other agents.

2.10 Genetic Algorithms

2.10.1 Metaphor

The Genetic Algorithms (GA) are like other Evolutionary Algorithms based on the metaphor of natural evolution or Darwinian evolution[12]. Evolution is based on three fundamental principles according to Darwin: replication, variation and natural selection [20]. In nature asexual reproduction by cell division is an example of replication, where one individual produces an identical copy of itself. However, this process can not go on in nature without any variation introduced by mutation of genes. Another type of variation is sexual reproduction where two individuals produce offspring which has a recombination of genetic material from the two parents. This can be described as crossover of genes or rather, crossover of chromosomes containing a set of genes. The probability that an individual will survive to reproduce itself can be described as the fitness, i.e. the higher fitness the higher probability of reproduction or *survival of the fittest*. The natural selection can be seen as an optimization procedure where the fitness is maximized.

2.10.2 Genetic Algorithms for Optimization of Continuous Functions

For combinatorial optimization the solution is often directly represented by a binary bit vector. The simplest way to introduce mutation is to flip a number of bits. Crossover is easily done by recombining two bit vectors in a way, that parts of the vector of the offspring is identical to one parent vector and the rest is identical to the bit vector of the other parent. For continuous variables this procedure has to be modified. The procedure described in the following is slightly modified version of the one used by [17] for optimization of econometric functions. Again a solution in the continuous search space is defined by the vector $\mathbf{x}$ where $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ and the x_i's are coordinates of real valued variables and n is the number of variables. However, the solution $\mathbf{x}$ is also represented by a binary bit vector $\mathbf{a}$ called an individual. The individual is a concatenation of all the genes a_i, i.e. $\mathbf{a} = a_1 a_2 \cdots a_n$. Thus, each gene a_i represents a solution coordinate x_i. The genes a_i are made of bits: $a_i = (a_{i,1}, a_{i,2}, \ldots, a_{i,m})$ where m is the number of bits[6]. The relation between the coordinate value x_i and the gene bit value a_i is determined by equation (2.5) where the coordinate is

[6] meaning the total number of bits in the bit vector $\mathbf{a}$ is equal to $n \cdot m$

constrained by $u_i \leq x_i \leq v_i$.

$$x_i = u_i + \frac{v_i - u_i}{2^l - 1} \sum_{j=1}^{m} a_{i,j} 2^{j-1} \tag{2.5}$$

This binary representation of the coordinate values result in a discretization of the search space. A way to make the discretization adaptive could be to adapt lower and upper limits (u_i and v_i). Intensification would be done by decreasing the gap $v_i - u_i$ at an interval of a number of generations.

The procedure of the Genetic Algorithm is outlined in pseudocode 2.10.1. The parameters described in the following are assumed initialized at the start. The initial population of λ individuals is defined in line 2–6 pseudocode 2.10.1. All bits of all λ new individuals $\mathbf{a}^{(k,t)}$ at time $t = 1$ are chosen at random with equal probability between zero and one, line 4. Until a stopping criterion is satisfied the generations continues to evolve, line 7. The stopping criterion could be a maximum number of iterations or a minimal improvement of the objective value $f(\mathbf{x}^{(k,t)})$ over a number of iterations. For each generation the individuals of the next population are selected, line 8. The generation counter is increased in line 9, meaning that the next generation becomes the current generation. Then the population is replaced by its offspring, line 10. Finally the members of the population have a risk of mutating, line 11. The procedure returns the solution with the lowest objective value considering all individuals in all generations, line 13. The output is not guaranteed locally optimal by the framework described here. Thus, a local optimization procedure as the Nelder-Mead simplex can be utilized for this purpose.

Pseudocode 2.10.1 Procedure GA

```
 1: t ← 1
 2: Population_t ← ∅
 3: for k = 1 to λ do
 4:     a^(k,t) ← random bit vector
 5:     Population_t ← Population_t ∪ {a^(k,t)}
 6: end for
 7: while Termination criterion not satisfied do
 8:     Population_{t+1} ← Selection(Population_t)
 9:     t ← t + 1
10:     Population_t ← Reproduction(Population_t)
11:     Population_t ← Mutation(Population_t)
12: end while
13: return x^(k,t)|{min f(x^(k,t)), ∀k, t}
```

2.10.2.1 Selection

The selection procedure is shown in pseudocode 2.10.2. The procedure runs through all λ individuals of the next population, line 1 pseudocode 2.10.2. The h'th individual $\mathbf{a}^{(h,t)}$ of the current population survives to become the k'th individual $\mathbf{a}^{(k,t+1)}$ of the new population at a probability $P_{\mathbf{a}^{(h,t)}}$ proportional to its relative fitness, line 5–6. The probability in the *fitness proportionate selection* is given by equation (2.6).

$$P_{\mathbf{a}^{(h,t)}} = \frac{-f(\mathbf{x}^{(h,t)}}{\sum_{l=1}^{\lambda} -f(\mathbf{x}^{(l,t)})} \tag{2.6}$$

Note that the fitness values of all individuals of a population has to be positive. If this is not the case the values should be shifted by adding a constant. The selection procedure allows for an individual to become a member of the new population several times. The higher the fitness the higher the probability of being represented more than once in the new population.

Pseudocode 2.10.2 Procedure Selection

```
1:  for k = 1 to λ do
2:      draw random χ from uniform interval [0; 1[
3:      h ← 1
4:      while a^(k,t+1) is not defined do
5:          if Σ_{l=1}^{h-1} −f(x^(l,t)) ≤ χ · Σ_{l=1}^{λ} −f(x^(l,t)) < Σ_{l=1}^{h} −f(x^(l,t)) then
6:              a^(k,t+1) ← a^(h,t)
7:          end if
8:          h ← h + 1
9:      end while
10: end for
```

2.10.2.2 Reproduction

The reproduction procedure within the Genetic Algorithm is presented in pseudocode 2.10.3. The individuals of the population are put into a random order and a set of offspring is initialized, line 1–2 pseudocode 2.10.3. Until the number of individuals amongst offspring |Offspring| is the same as the population size λ new offspring are produced, line 4. A pair of parents $\alpha = \mathbf{a}^{(k \ mod \ \lambda,t)}$ and $\alpha = \mathbf{a}^{((k+1) \ mod \ \lambda,t)}$ are chosen from the population at a probability $P_{\text{crossover}}$ for increasing k, line 6–8. The pairing can be seen as random since the ordering of the population is random. A new offspring is produced by the crossover operator on the parents α and β, line 9. The crossover operator produces two offspring as described by table 2.1 and returns one of them with equal probability. The parents are shown in the first row and the offspring in the second.

The two offspring are produced by a one-point crossover procedure, i.e. by cutting both parents' concatenated string of genes at a position *pos* and putting together the first *pos* bits from parent α with the last $n \cdot m - pos$ bits from parent β and vice versa. The integer *pos* is chosen at random between 1 and $n \cdot m$. When enough offspring is produced the current population is replaced by its offspring, i.e. generational replacement where no parents survive, line 13 pseudocode 2.10.3.

Pseudocode 2.10.3 Procedure Reproduction

```
 1: order Population_t at random
 2: Offspring ← ∅
 3: k ← 1
 4: while |Offspring| < λ do
 5:    draw random χ from uniform interval [0; 1]
 6:    if χ < P_crossover then
 7:       α ← a^(k mod λ,t)
 8:       β ← a^((k+1) mod λ,t)
 9:       Offspring ← Offspring ∪ {Crossover(α, β)}
10:    end if
11:    k ← k + 1
12: end while
13: Population_t ← Offspring
```

First Individual

$\alpha_{1\cdot1}$	$\alpha_{1\cdot2}$	...	α_{pos}	α_{pos+1}	...	$\alpha_{n\cdot m}$

Second Individual

$\beta_{1\cdot1}$	$\beta_{1\cdot2}$	...	β_{pos}	β_{pos+1}	...	$\beta_{n\cdot m}$

$\alpha_{1\cdot1}$	$\alpha_{1\cdot2}$	...	α_{pos}	β_{pos+1}	...	$\beta_{n\cdot m}$

$\beta_{1\cdot1}$	$\beta_{1\cdot2}$	...	β_{pos}	α_{pos+1}	...	$\alpha_{n\cdot m}$

Table 2.1: The Crossover procedure

2.10.2.3 Mutation

The mutation procedure is described in pseudocode 2.10.4. The procedure is basically running through all $n \cdot m$ bits of all λ individuals, line 1 and 2 in pseudocode 2.10.4. By a low probability P_{mutation} the bit is mutated, i.e. flipped to the opposite binary bit, line 5.

2.10.3 Variations of Genetic Algorithms

The main difference in Genetic Algorithm approaches to optimization lie in the sub procedures. The strategy described in section 2.10.2.1 - for selecting parents

Pseudocode 2.10.4 Procedure Mutation

```
1: for k = 1 to λ do
2:    for g = 1 to n · m do
3:       draw random χ from uniform interval [0; 1]
4:       if χ < P_mutation then
5:          FlipBit a_g^(k,t)
6:       end if
7:    end for
8: end for
```

to reproduce - is based on a fitness proportionate selection. This resembles the way to select directions in the ant colony optimization in section 2.8.2 page 22. When the variance of fitness values is small the fitness proportionate selection is close to random. Other strategies avoiding this are tournament selection and rank based selection [20]. In tournament selection the individual with the highest fitness is chosen from a pool of a number of randomly selected individuals. In rank based selection the individual $\mathbf{a}^k$ is chosen according to its rank k in the population, i.e. the population is ordered for increasing fitness.

The parents are replaced completely by their offspring in the procedure in section 2.10.2.2, i.e. generational replacement. Another way to replace parents is the steady state selection [20] where only some of the parents are replaced by a number of offspring lower than the population size. This can be the worst parents, the oldest parents or randomly chosen parents. On the other hand, parents can be allowed to compete with the offspring in a temporary pool of both offspring and parents.

As mentioned in section 2.10.2.2 the crossover operator used is a one-point crossover. Alternatives are a two-point crossover or a generalized k-point crossover where the bit strings are cut at k different positions chosen at random. The offspring is combined by assembling parts from the two cut parents in turn.

An alternative to the bit flip mutation suggested in section 2.10.2.3 is the inversion operator [20]. Inversion is applied by choosing a bit substring of the individual at random. The substring is then reversed in order to mutate the individual. This kind of mutation could change a single individual a lot if applied whereas, the bit flip mutation is only likely to change an individual a lot if the probability of mutation is high. On the other hand, if probability of mutation is low the inversion operator is likely to leave the individual unchanged.

2.10.4 Pros and Cons of Genetic Algorithms

The main problem with the Genetic Algorithm described here is the discretization of the search space. Depending on the structure this could shut of access to many locally and maybe globally optimal solutions. Usually it is important for a metaheuristic to be able to search the whole search space. A solution to this problem is to make the discretization adaptive in some way, as suggested earlier. Comparing to many other procedures there is an extra step of going from the individual - represented by a bit string - to the solution - represented by a vector in the continuous search space - before calculating the objective value. If this detour is costly in computational time is a question perhaps to be answered by testing. Since the Genetic Algorithms are not neighborhood based diversity is easily obtained by the crossover and mutation operators. However, the lack of ability to search a local neighborhood for better solutions means that good local optima could be overlooked even if these are close to individuals in the population.

2.11 Evolution Strategies

2.11.1 Metaphor

The metaphor of Evolution Strategies (ES) is closely related to that of Genetic Algorithms in section 2.10.1 page 25. The difference lie in the representation of an individual, population, parent and offspring as described below.

2.11.2 Evolution Strategies for Optimization of Continuous Functions

Unlike the Genetic Algorithms the individuals in the Evolution Strategies are represented by floating point vectors. Thus, the individual can directly represent a solution $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ in the continuous search space, where n is the number of variables. The variables of the solution can be interpreted as the genes of the individual. Each variable x_i is constrained by $u_i \leq x_i \leq v_i$. The k'th individual in a population is combined by a pair of vectors $(\mathbf{x}^{(k,t)}, \sigma^{(k,t)})$, where $\mathbf{x}^{(k,t)}$ is a solution in the search space at time t and $\sigma^{(k,t)}$ is a vector of standard deviations at time t. That means $x_i^{(k,t)}$ one type of genes whereas $\sigma_i^{(k,t)}$ is another.

The framework describing the Evolution Strategies is very similar to that of Genetic Algorithms. Only, the selection procedure is made after creating offspring and letting them mutate. In contradiction to the Genetic Algorithms individuals are not selected to produce offspring, with a probability according to their fitness. The Evolution Strategies procedure is seen in pseudocode 2.11.1. The initial population is chosen at random, line 3–11 pseudocode 2.11.1. Each gene x_i^k of the k'th individual is set at random between an upper v_i and lower bound u_i, line 5–6. The deviation is likewise initialized at random in the interval between zero and $\sigma_{\max}$, line 7–8. The main loop of the Evolution Strategies runs until a stopping criterion is satisfied. This could be a minimum improvement ϵ of the objective value of the best individual over two generations, i.e. $f(\mathbf{x}_{\text{best}}^{(k,t-1)}) - f(\mathbf{x}_{\text{best}}^{(k,t)}) < \epsilon$, a maximum number of iterations or both. The genetic evolution is repeatedly utilizing reproduction of a population followed by mutation and finally a selection procedure, line 13–15. At the termination of the algorithm the solution with the lowest objective value is returned. It is only necessary to regard the current iteration when looking for the fittest individual due to the selection procedure which is described later in section 2.11.2.3 and 2.11.3.

Pseudocode 2.11.1 Procedure ES

1: $t \leftarrow 1$
2: $\text{Population}_t \leftarrow \varnothing$
3: **for** $k = 1$ to μ **do**
4: **for** $i = 1$ to n **do**
5: draw random χ from uniform interval $[0; 1]$
6: $x_i^{(k,t)} \leftarrow u_i + \chi(v_i - u_i)$
7: draw random ϕ from uniform interval $[0; 1]$
8: $\sigma_i^{(k,t)} \leftarrow \phi \cdot \sigma_{\max}$
9: **end for**
10: $\text{Population}_t \leftarrow \text{Population}_t \cup \{(\mathbf{x}^{(k,t)}, \sigma^{(k,t)})\}$
11: **end for**
12: **while** Termination criterion **not** satisfied **do**
13: $\text{Offspring}_t \leftarrow \text{Reproduction}(\text{Population}_t)$
14: $\text{Offspring}_t \leftarrow \text{Mutation}(\text{Offspring}_t)$
15: $\text{Population}_{t+1} \leftarrow \text{Selection}(\text{Population}_t, \text{Offspring}_t)$
16: $t \leftarrow t + 1$
17: **end while**
18: **return** $\mathbf{x}^{(k,t)} | \{\min f(\mathbf{x}^{(k,t)}), \forall k\}$

2.11.2.1 Reproduction

The reproduction procedure is given by pseudocode 2.11.2. The reproduction procedure works by creating λ new offspring, line 2 pseudocode 2.11.2. An offspring is produced by first choosing two parents at random from the current population, line 5. All n pair of genes are crossed over between the two parents k and m. Each pair of genes γ_i^l is taken from either of the parents at an equal probability of 50%, line 6–9. The offspring γ^l is inserted into the population of offspring, line 12. An alternative crossover operator is given by table 2.2. Instead of making a clone of a pair of genes in one parent this operator recombines the genes. As seen from the last line of table 2.2 the genes of the offspring is the average values of the genes of the two parents.

Pseudocode 2.11.2 Procedure Reproduction

```
 1: Offspring_t ← ∅
 2: for l = 1 to λ do
 3:    draw two integers k and m at random from interval {1, 2, ..., μ}
 4:    for i to n do
 5:       draw random χ from uniform interval [0; 1[
 6:       if χ < 0.5 then
 7:          γ_i^l ← ((x_i^(k,t))(σ_i^(k,t)))
 8:       else
 9:          γ_i^l ← ((x_i^(m,t))(σ_i^(m,t)))
10:       end if
11:    end for
12:    Offspring_t ← Offspring_t ∪ {γ^l}
13: end for
```

First Parent	Second Parent
$\Big((x_1^k, x_2^k, \ldots, x_n^k), (\sigma_1^k, \sigma_2^k, \ldots, x_n^k)\Big)$	$\Big((x_1^m, x_2^m, \ldots, x_n^m), (\sigma_1^m, \sigma_2^m, \ldots, x_n^m)\Big)$
$\Big(((x_1^k + x_1^m)/2, (x_2^k + x_2^m)/2, \ldots, (x_n^k + x_n^m)/2), ((\sigma_1^k + \sigma_1^m)/2, (\sigma_2^k + \sigma_2^m)/2, \ldots, (\sigma_n^k + \sigma_n^m)/2)\Big)$	

Table 2.2: Alternative crossover procedure for ES

2.11.2.2 Mutation

The vector of standard deviations $\sigma^{(k,t)}$ is used in the mutation procedure given by pseudocode 2.11.3. Each pair of genes in each individual amongst all the offspring is mutated, line 1–2 pseudocode 2.11.3. At first the i'th gene - related

to the standard deviation - is mutated according to line 3. The exponential factor can be seen as a way to intensify the search procedure where the τ' and τ are depending on the procedure. Secondly, the i'th gene related to the value of the solution variable is mutated by adding a normally distributed random number with expectation zero and mean σ_i', line 4. The mutation of the standard deviations is an option which can be used to control the relative "amount" of mutation of the solution coordinate. On the other hand, if the "amount" of mutation applied by the operator should be the same throughout the algorithm one can decide only to apply mutation to the genes related to the value of the solution variable. Contrary to the Genetic Algorithms a low probability P_{mutation} is not defined in the standard Evolution Strategies. This is due to the fact that mutation is the main source of diversity in Evolution Strategies [17].

Pseudocode 2.11.3 Procedure Mutation

1: **for** $l = 1$ to λ **do**
2: **for** $i = 1$ to n **do**
3: $\sigma_i' \leftarrow \sigma_i^{(l,t)} \exp(\tau' N(0,1) + \tau N_i(0,1))$
4: $x_i^{(l,t)} \leftarrow x_i^{(l,t)} + N(0, \sigma_i')$
5: **end for**
6: **end for**

2.11.2.3 Selection

The last part of the Evolution Strategies is the selection of individuals to survive to the next generation. All parents in the population and all offspring are competing on an equal basis for survival. The selection procedure is given by pseudocode 2.11.4. The selection chooses μ individuals for survival, line 4. At all time the best individual - with the lowest objective value - is chosen from the temporary population of both parents and offspring, line 5. The individual is removed from the temporary population and inserted into the surviving population, line 6–7.

Pseudocode 2.11.4 Procedure Selection

1: $\text{Population}_{temp} \leftarrow \text{Population}_t \cup \text{Offspring}_t$
2: $\text{Population}_{t+1} \leftarrow \varnothing$
3: Let $\beta \in \text{Population}_{temp}$
4: **for** $k = 1$ to μ **do**
5: $(\mathbf{x}^{(k,t)}, \sigma^{(k,t)}) \leftarrow \beta | \{\min f(\beta), \forall \beta \in \text{Population}_{temp}\}$
6: $\text{Population}_{temp} \leftarrow \text{Population}_{temp} \backslash \{(\mathbf{x}^{(k,t)}, \sigma^{(k,t)})\}$
7: $\text{Population}_{t+1} \leftarrow \text{Population}_{t+1} \cup \{(\mathbf{x}^{(k,t)}, \sigma^{(k,t)})\}$
8: **end for**

2.11.3 Variations of Evolution Strategies

As for the Genetic Algorithms the variations within Evolution Strategies lie in the different sub procedures. An alternative crossover operator is already described for the reproduction procedure in section 2.11.2.1. As described in section 2.11.2.2 the mutation operator can be applied to only the values of the solution variables instead of both to solution variables and standard deviations. Another possibility is to use the exponential function on an alternative function than the $\tau' N(0,1) + \tau N(0,1)$. Various possibilities exist when it comes to the selection strategies. The strategy in section 2.11.2.3 can be described as a $(\lambda + \mu)$–strategy which means that the μ fittest survivors for reproduction are chosen from a temporary population of the λ offspring and μ parents all together. A direct consequence of this selection strategy is the possibility for a very fit individual to survive for several generations on the expense of some offspring. An alternative selection strategy is the (λ, μ)–strategy where the μ fittest survivors for reproduction is chosen amongst the λ offspring. For this to work the number of offspring has to be strictly larger than the number of parents in the population[7]. The (λ, μ)–strategy was used in the Genetic Algorithm described in section 2.10.2.1. As for other strategies these can be combinations of the two opposite strategies described above. A compromise could be to let parents compete with the generation of their own children only or an upper bound could be set on the number of parents allowed to survive each generation.

2.11.4 Pros and Cons of Evolution Strategies

An obvious strength of the Evolution Strategies for continuous functions is that it is easily utilized. No detours have to be made in order to adapt Evolution Strategies to this field of optimization in contrary to many other metaheuristics developed for combinatorial optimization. In fact, the Evolution Strategies were first developed for this kind of optimization of continuous variables by Rechenberg and Schwefel according to [20]. Comparing the Genetic Algorithms and the Evolution Strategies it is seen that the Genetic Algorithms are easily applied to the combinatorial nature of some problems whereas, the Evolution Strategies are easily applied to problems that are defined over continuous variables. However, this does not directly imply that either one is faster at finding good close-to-optimal solutions to some problems but according to [17] the Genetic Algorithms were performing worst on optimization of econometric functions compared to simulated annealing and evolutions strategies.

[7] for $\lambda < \mu$ the population size would decrease and for $\lambda = \mu$ there would be no fitness based selection

2.12 Memetic Algorithms

2.12.1 Metaphor

The metaphor of Memetic Algorithms (MA) is somehow related to that of the Genetic Algorithms and Evolution Strategies. However, a number of important differences exist. The Memetic Algorithms are based on the concept of cultural evolution contrary to that of genetic evolution. The meaning of cultural evolution is that cultural information is transmitted in an evolutionary process. The objects to hold this information are called memes which is short for the Greek word *mimeme* [20]. Memes can be viewed as ideas, skills or ways of doing things and are transmitted to other individuals by imitation. In this way cultural evolution is much faster than the process of genetic evolution. Perhaps the most crucial difference between the genetic evolution and the cultural evolution is that in the latter an individual have the possibility of recombining memes and introducing an innovative component before passing the memes on to others. From an optimization point of view the innovation can be introduced by some sort of improvement heuristic such as local search or Descent Methods. With this in mind a Memetic Algorithm can be viewed as a hybrid Genetic Algorithms utilizing local search [8, 20]. Comparing to genetic evolution the cultural evolution is a goal oriented process where each individual acts consciously by doing improvements.

2.12.2 Memetic Algorithms for Optimization of Continuous Functions

The idea of the Memetic Algorithm described in this section comes from the procedure presented by [20]. However, due to the fact that Merz optimize combinatorial problems the sub procedures are carried out differently. In fact notation and sub procedures used in the following are closely related to that of Evolution Strategies described in section 2.11. The difference is that the components of an individual $\mathbf{x}^{(k,t)}$ are regarded as memes not genes, i.e. the memes are given by $x_i^{(k,t)}$ where $1 \leq i \leq n$ and n is the number of memes and t is the time or generation number. The index k describes the place of the individual in the population and is bounded by the number of individuals μ in the population.

The procedure for the Memetic Algorithm is given by pseudocode 2.12.1. The lower and upper limits u_i and v_i are assumed to be initialized together with the maximum standard deviation $\sigma_{\max}$ at the start of the procedure. The initial

population is created almost in the same way as for the Evolution Strategies except from that a local search is applied to each individual before inserting it into the initial population, line 10–11 pseudocode 2.12.1. For all μ individuals each n memes $x_i^{(k,t)}$ is set at random in the search space, line 4–6. As for now the local search procedure can be thought of as the Descent Method from section 2.3.2. The Memetic Algorithm continue to evolve the population until a stopping criterion is satisfied, line 11. As for Evolution Strategies this criterion could be related to a minimum improvement of the objective value or a maximum number of iterations. A reproduction procedure is used in each generation, line 14. Mutation is allowed over a temporary population of both the current population and the offspring from reproduction, line 16. Each iteration is concluded by selecting the population of the next generation, line 18. The selection of survivors is done amongst the current population, its offspring from reproduction and the mutated temporary population, line 17. Finally the individual with the lowest objective value in the final population is returned, line 21. Because of the selection strategy described in section 2.12.2.3 the fittest individual is to be found in the final population.

Pseudocode 2.12.1 Procedure MA

1: $t \leftarrow 1$
2: $\text{Population}_t \leftarrow \varnothing$
3: **for** $k = 1$ to μ **do**
4: **for** $i = 1$ to n **do**
5: draw random χ from uniform interval $[0; 1]$
6: $x_i^{(k,t)} \leftarrow u_i + \chi(v_i - u_i)$
7: draw random ϕ from uniform interval $[0; 1]$
8: $\sigma_i^{(k,t)} \leftarrow \phi \cdot \sigma_{\max}$
9: **end for**
10: $\mathbf{x}^{(k,t)} \leftarrow \text{LocalSearch}(\mathbf{x}^{(k,t)}, \sigma^{(k,t)})$
11: $\text{Population}_t \leftarrow \text{Population}_t \cup \{(\mathbf{x}^{(k,t)}, \sigma^{(k,t)})\}$
12: **end for**
13: **while** Termination criterion **not** satisfied **do**
14: $\text{Offspring}_t \leftarrow \text{Reproduction}(\text{Population}_t)$
15: $\text{Population}_{\text{temp}} \leftarrow \text{Population}_t \cup \text{Offspring}_t$
16: $\text{Population}_{\text{mutated},t} \leftarrow \text{Mutation}(\text{Population}_{\text{temp}})$
17: $\text{Population}_{\text{temp}} \leftarrow \text{Population}_t \cup \text{Offspring}_t \cup \text{Population}_{\text{mutated},t}$
18: $\text{Population}_{t+1} \leftarrow \text{Selection}(\text{Population}_{\text{temp}})$
19: $t \leftarrow t + 1$
20: **end while**
21: **return** $\mathbf{x}^{(k,t)} | \{\min f(\mathbf{x}^{(k,t)}), \forall k\}$

In the Memetic Algorithm by [20] the use of a meta-mutation is suggested in order to update the population after selection, in case the population is converged. The meta-mutation takes care of mutating all individuals except the best one in the population. The procedure can be seen as a way to diversify

the search after the population has automatically converged. The reason meta-mutation is not described in pseudocode 2.12.1 is the lack of evidence of the automatic convergence of a population based on continuous variables. After all the convergence is highly dependant on the local search procedure. Furthermore [8] who use Memetic Algorithms for optimization of continuous functions do not describe a similar meta-mutation. Thus, the meta-mutation is left as an open question to be answered by testing of different versions of Memetic Algorithms.

2.12.2.1 Reproduction

The reproduction procedure of the Memetic Algorithm is given by pseudocode 2.12.2. It is closely related to that of Evolution Strategies with the important difference of applying local search to each offspring γ^l before it is accepted to the population of offspring, line 12-13 pseudocode 2.12.2. The number of offspring to be created is denoted $\lambda_{\text{reproduce}}$, line 2. Exactly like the Evolution Strategies the offspring is made by crossover of two parents chosen at random, line 3. All memes γ_i^l are taken from either one of the parents by equal probability, line 5–9. Again the alternative crossover operator defined by table 2.3 can be used

Pseudocode 2.12.2 Procedure Reproduction

```
 1: Offspring_t ← ∅
 2: for l = 1 to λ_reproduce do
 3:    draw two integers k and m at random from interval {1, 2, ..., μ}
 4:    for i to n do
 5:       draw random χ from uniform interval [0; 1[
 6:       if χ < 0.5 then
 7:          γ_i^l ← ((x_i^(k,t))(σ_i^(k,t)))
 8:       else
 9:          γ_i^l ← ((x_i^(m,t))(σ_i^(m,t)))
10:       end if
11:    end for
12:    γ^l ←LocalSearch(γ^l)
13:    Offspring_t ← Offspring_t ∪ {γ^l}
14: end for
```

instead. The memes are chosen as the average of the two memes of the two parents.

First Parent	Second Parent
$\Big((x_1^k, x_2^k, \ldots, x_n^k), (\sigma_1^k, \sigma_2^k, \ldots, x_n^k)\Big)$	$\Big((x_1^m, x_2^m, \ldots, x_n^m), (\sigma_1^m, \sigma_2^m, \ldots, x_n^m)\Big)$
$\Big(\big((x_1^k + x_1^m)/2, (x_2^k + x_2^m)/2, \ldots, (x_n^k + x_n^m)/2\big), \big((\sigma_1^k + \sigma_1^m)/2, (\sigma_2^k + \sigma_2^m)/2, \ldots, (\sigma_n^k + \sigma_n^m)/2\big)\Big)$	

Table 2.3: Alternative crossover procedure for MA

2.12.2.2 Mutation

The mutation procedure for Memetic Algorithms is defined by pseudocode 2.12.3. The mutation operator is applied to λ_{mutate} individuals, line 1 pseudocode 2.12.3. The integer m is chosen at random between one and $\mu+\lambda_{\text{reproduce}}$ which is the number of individuals in the temporary population consisting of both parents and offspring. Each memes of the individual $\mathbf{x}^{(m,t)}$ in the temporary population is added a normally distributed random number with the standard deviation $\sigma_i^{(m,t)}$ and the expected value zero, line 4. Local search is applied to the resulting individual before it is added as a new individual β^l to the mutated population, line 6–7. Note that the standard deviation is not self adapted by default as for Evolution Strategies. Thus, the standard deviation for the new individual is inherited from the mutated individual, line 6.

Pseudocode 2.12.3 Procedure Mutation

1: **for** $l = 1$ to λ_{mutate} **do**
2: draw an integer m at random from interval $\{1, 2, \ldots, \mu + \lambda_{\text{reproduce}}\}$
3: **for** $i = 1$ to n **do**
4: $x_i^{(l,t)} \leftarrow x_i^{(m,t)} + N(0, \sigma_i^{(m,t)})$
5: **end for**
6: $\beta^l \leftarrow \text{LocalSearch}\big((\mathbf{x}^{(l,t)}), (\sigma^{(m,t)})\big)$
7: $\text{Population}_{\text{mutated},t} \leftarrow \text{Population}_{\text{mutated},t} \cup (\beta^l)$
8: **end for**

2.12.2.3 Selection

The selection procedure given by pseudocode 2.12.4 does not differ from the one suggested for Evolution Strategies. For Memetic Algorithms the temporary population contains the current population, the offspring created by recombination and local search together with the mutated population which is also exposed to local search. Thus, the number of individuals in the temporary population used for selection becomes $\mu + \lambda_{reproduce} + \lambda_{mutate}$. Individuals for the

next population is selected on the basis of their fitness. This implies that the μ best individuals are put into the population of the next generation, line 3–7 pseudocode 2.12.4. The procedure works by repeatedly selecting the individual with the minimum objective value from the temporary population, line 4. Then the individual is removed from the temporary population and inserted into the next generation, line 5–6. This selection is a $(\lambda + \mu)$-strategy where λ is equal to $\lambda_{reproduce} + \lambda_{mutate}$. The parents are competing on equal basis with the offspring produced by recombination, mutation and possibly both.

Pseudocode 2.12.4 Procedure Selection

1: $\text{Population}_{t+1} \leftarrow \varnothing$
2: Let $\alpha \in \text{Population}_{temp}$
3: **for** $k = 1$ to μ **do**
4: $\quad (\mathbf{x}^{(k,t)}, \sigma^{(k,t)}) \leftarrow \alpha | \{\min f(\alpha), \forall \alpha \in \text{Population}_{temp}\}$
5: $\quad \text{Population}_{temp} \leftarrow \text{Population}_{temp} \backslash \{(\mathbf{x}^{(k.t)}, \sigma^{(k,t)})\}$
6: $\quad \text{Population}_{t+1} \leftarrow \text{Population}_{t+1} \cup \{(\mathbf{x}^{(k,t)}, \sigma^{(k,t)})\}$
7: **end for**

2.12.3 Variations of Memetic Algorithms

The possibility of using a meta-mutation procedure after conversion of the population is described in section 2.12.2. It deserves to be mentioned again as a possible variant of Memetic Algorithms.

The local search suggested in the Memetic Algorithm could resemble the Descent Methods described in section 2.3.2. However, the local search procedure is utilized at initialization and throughout both the reproduction and mutation procedure at each iteration. Thus, it is desirable to make it fast. Because the Descent Method is time consuming in this sense a simpler version of local search is likely to perform better. A simple and fast local search could use the framework of the Descent Method but terminate as soon as an improved solution is found. In fact this framework is suggested by [8].

As described in the previous sections variations of Genetic Algorithms and Evolution Strategies could be based on different selection strategies. Likewise, Memetic Algorithms can utilize different selection strategies or combinations of these.

2.12.4 Pros and Cons of Memetic Algorithms

The most obvious advantage of the Memetic Algorithms is the combination of nice properties from both traditional Evolutionary Algorithms and point-to-point neighborhood heuristics. At the same time the Memetic Algorithms get rid of the compromising drawback of the traditional Evolutionary Algorithms; namely the lack of ability to look for a better solution in the local neighborhood of an individual.

A negative property of Memetic Algorithms could be the relatively time consuming loops consisting of both reproduction and local search together with mutation and local search.

2.13 Iterated Local Search

2.13.1 Metaphor

The Iterated Local Search (ILS) is based on two metaphors. The local search used in Iterated Local Search is based on the hill climbing metaphor. It resembles any other local search or descent method that will terminate in a local optima. The local search procedure is iterated. Rather than choosing the starting points at random for each new iteration the starting point is found by the mutation operator known from other Evolutionary Algorithms. However, the Iterated Local Search cannot be described as a population based method since the algorithm operates on a single solution only, i.e. a single agent approach.

The Iterated Local Search can be viewed as a special instance of the Memetic Algorithm. The Memetic Algorithm can act as an Iterated Local Search when the population size is equal to one and the number of offspring to be reproduced $\lambda_{\text{reproduce}}$ is equal to zero [20].

2.13.2 Iterated Local Search for Optimizing Continuous Functions

Since the Iterated Local Search is a special case of the Memetic Algorithm the pseudocode is also similar. The procedure is given by pseudocode 2.13.1. The initial solution is created in line 3–6, pseudocode 2.13.1. The box constraints given by u_i and v_i respectively are set prior to the procedure. The steps of

mutation and local search are repeated until a stopping criterion is satisfied, line 9. The mutation process of the Iterated Local Search is similar to that of Evolution Strategies given by pseudocode 2.11.3 on page 33. The only difference is that the mutation works on a single individual instead of a whole population. The local search sub procedure also works on an individual solution and is similar to any local search. The new solution - generated by mutation and local search - is only kept until the next generation if the objective value is improved, line 10–13. Otherwise, the current solution is kept until the next generation, line 14–15. At the end the best solution is returned, line 19.

Pseudocode 2.13.1 Procedure ILS

```
 1: t ← 1
 2: for i = 0 to n do
 3:    draw random χ from uniform interval [0; 1]
 4:    x_i^t ← u_i + χ(v_i − u_i)
 5:    draw random φ from uniform interval [0; 1]
 6:    σ_i^t ← φ · σ_max
 7: end for
 8: x^t ← LocalSearch(x^t, σ^t)
 9: while termination criterion not satisfied do
10:    (x', σ') ← Mutate(x^t, σ^t)
11:    x' ← LocalSearch(x^t, σ^t)
12:    if f(x') < f(x^t) then
13:       x^{t+1} ← (x', σ')
14:    else
15:       x^{t+1} ← (x^t, σ^t)
16:    end if
17:    t ← t + 1
18: end while
19: return x^t
```

2.13.3 Variations of Iterated Local Search

Since the Iterated Local Search can already be seen as a special case of the Memetic Algorithm not many changes can be applied within the relatively narrow framework. However, the initialization of the standard deviation σ_i could be fixed to a number $\sigma_{\max}$ instead of being random. This would make sense because there is only one individual and it makes the standard deviation more directly controllable by the parameter $\sigma_{\max}$.

2.13.4 Pros and Cons of Iterated Local Search

The Iterated Local Search is more dependent on the starting point than a population based algorithm since a population based approach is more likely to explore the search space more due to the multiple starting points. Depending on the problem structure this drawback could have more or less serious consequences. What makes the Iterated Local Search interesting is the lower time consumption than the general Memetic Algorithm. By limiting the local search to few descending steps the sub procedure becomes fast. The only operator applied for each iteration is the mutation which is also fast. Therefore, the Iterated Local Search can perform very fast iterations.

It is hard to say whether the Iterated Local Search will converge enough. The mutation operator can be seen as a way of defining a neighborhood in the continuous search space given by the standard deviation. This resembles very much the definition of neighborhood in the Descent Methods and Simulated Annealing. However, for combinatorial problems a local search would choose the best neighbor whereas a bit flip mutation would still be random. Thus, it makes more sense to differ between mutation and local search for combinatorial problems than for continuous functions. A possible way around this could be to restrict the local search to a Nelder–Mead algorithm or to allow any mutation although the offspring turns out to be a less fit individual than the parent.

Chapter 3

Comparing and Choosing Metaheuristic

3.1 Summary

This chapter deals with the comparison of the metaheuristics and the selection of one for implementation. The comparison is based on the description in the previous chapter focusing on how the metaheuristics are adapted to optimization of continuous functions. To aid the selection process a modified SWOT analysis is made for each of the methods. The steps carried out in this chapter are a convergent part of the overall solution process.

3.2 Comparison Scheme of the Metaheuristics

The comparison scheme given by table 3.1 should give an overview of how well the metaheuristics are adapted to continuous function optimization. The statements are implicitly written in the chapter describing the metaheuristics and are based on various papers given in the bibliography. However, table 3.1 is an attempt to quantify the properties to make way for direct comparison. The first line illustrates whether the metaheuristic take advantage of a point-to-point im-

provement heuristic. Only the Genetic Algorithm and the Evolution Strategies do not use neighborhood search. Second, it is seen that the Genetic Algorithm, Evolution Strategies, Memetic Algorithm and Iterated Local Search uses the property to evolve populations over generations. In the third line the Nelder–Mead simplex is shown to be the only method to use the geometrical properties of the solution points in the search space. The Nelder–Mead procedure moves each new point in the direction where a possible improvement of the objective value could be expected. The idea can be compared to interpolation and extrapolation. A judgement of how well the metaheuristic is suited for optimization of continuous functions is given in line four. Each method is ranked from one to three. The value three means that the metaheuristic is either developed for continuous optimization directly or it is extremely easy to adapt. A ranking of two is somewhat intermediate but can be adapted quite easily. If the heuristic only has a value of one it reflects that the method has little to do with the original metaheuristic for combinatorial problems and is only adapted by applying many extra sub-procedures. It can be questioned whether the actual goal of developing such methods is to find the best solution easily or just to adapt a certain predefined heuristic to the area of continuous functions optimization. Line five presents a somewhat subjective judgement of the quality of the different descriptions given in different papers. The ranking is based on a study of the literature in the bibliography. The last line states whether the metaheuristic is able to escape a local minimum. Only the Descent Methods and the Nelder–Mead simplex are unable of that.

	Metaheuristics								
	DM	SA	NM	TS	AC	GA	ES	MA	ILS
1 Utilization of neighborhood search	√	√	√	√	√			√	√
2 Utilization of evolutionary evolvement						√	√	√	√
3 Utilization of geometrical simplex		√							
4 Simplicity in adaption to continuous functions	3	3	3	1	1	2	3	3	3
5 Quality of the descriptions in other literature	?	3	3	2	1	3	3	2	?
6 Ability to escape local minimum		√		√	√	√	√	√	√

Table 3.1: Comparison scheme of the nine metaheuristics

3.3 SWOT Analysis

The comparison scheme presented in table 3.1 might provide an overview of the basic properties of the metaheuristics. However, it says little about the actual advantages and drawbacks of each metaheuristic. To give a more in dept analysis of the methods, a modified SWOT analysis is made for each metaheuristic. The

modification consists of emphasizing the *opportunities* with a main focus on *possibilities* of guiding the search procedure in a divergent way. The primary goal of the SWOT analysis is to assist in selecting the metaheuristic with the best possibilities of implementing a divergent/convergent search strategy. However, a secondary goal is to find other opportunities, threats, strengths and weaknesses and combine or solve these. The reason for the modification is that the selection of a metaheuristic should mainly rely on the number of opportunities and their character, such as the strengths and weaknesses are highly individual to each procedure and does not call for direct comparison.

3.3.1 Descent Methods

The first SWOT analysis given by table 3.2 considers the Descent Methods. As a direct search method the procedure does not have many opportunities of using divergent thinking. However, it can be used as a fast way of improving solutions developed by other metaheuristics. Note that the Nelder–Mead simplex - which is another direct search method - is not presented in a SWOT matrix. It can be assumed to have some of the same strengths, weaknesses, opportunities and threats as the Descent Methods. However, it might be a stronger procedure to find a pure local minimum because of the geometrical simplex properties.

3.3.2 Simulated Annealing

The Simulated Annealing presented in table 3.3 uses a somewhat simple framework. It is definitely worth using the Simulated Annealing to optimize continuous functions because of its simplicity, capability of escaping local minima and the overall convergence. However, it does not provide many tools for *diversification*[1] and *intensification*.

3.3.3 Tabu Search

The Tabu Search is seen in various version. As presented by the SWOT analysis in table 3.4 the search can be quite diverse, but this method also has a limited amount of possible tools for diversification/intensification.

[1] see section 4.3 for the connection between diversification/intensification and divergent/convergent thinking

Descent Methods	*Opportunities* - use for improving other metaheuristics - easy to combine with other metaheuristics	*Threats* - few possible variations in the simple framework - getting trapped in local minimum - can choose poor neighborhood solution when better exists no "intelligent" way of continuing search from local minimum
Strengths - fast iterations - relatively fast convergence - simple	- possible to implement as part of other metaheuristics utilize few iterations for fast improvement of solution	- let other methods handle the "intelligent" search approach
Weaknesses - cannot escape local minima - few ways to control diversification - diversification does not make sense because DM cannot escape local optima		-utilize random restart

Table 3.2: SWOT matrix on Descent Methods

Simulated Annealing	*Opportunities*	*Threats*
	- Parameters T and d can control diversification/intensification - use of local search in combination with other metaheuristics	- requires more running time than DM to perform better than DM - can accept ascent move when descent move is possible - can accept poor descent move when better is possible - if only a few iterations of some neighborhood based search is desired, one is better of with DM
Strengths - fast iterations - convergence is proofed - able to escape local minimum - tested on continuous functions	- Provide good results as stand-alone metaheuristic	- useful for functions with multiple optima obtains good solution relatively fast
Weaknesses - returns best solution slower than DM		- can provide good solutions without being combined with other metaheuristics

Table 3.3: SWOT matrix on Simulated Annealing

Tabu Search	*Opportunities* - very diverse search - several versions exists for continuous problems - hyper rectangles allow for intensification/diversification	*Threats* - does not exclude tabu solution forever exclude candidates close to tabu solution - possibly exclude global or local minimum
Strengths - memory to save good and bad solutions - promising list - does not get trapped in local minimum	- local search could be applied from all the solutions in the promising list	- might not find close to optimal solution but can find several good starting points for local search
Weaknesses - use and updating of memory - does not return local minimum	- use local search on one or more of the most promising points	- possibility of finding good solution in previously excluded part of search space

Table 3.4: SWOT matrix on Tabu Search

3.3.4 Ant Colony Optimization

The Ant Colony presented by table 3.5 is an intriguing metaphor, but the versions provided for continuous functions have obvious drawbacks. Furthermore, practically no tools are seen to guide the search in either divergent or convergent ways.

Ant Colony	*Opportunities* - speed of MAS can be improved by parallel programming - hybridization in different ways	*Threats* - most versions for continuous functions are poorly described in papers - large effort spend on less promising neighborhoods before convergence - ants are highly dependant on the choices of previous generations
Strengths MAS ⇒ diversity - ants highly dependant on the choices of previous generations	- good performance when parallelized - high probability of reaching good local optima	- will converge to the best local minimum found
Weaknesses - relatively slow convergence to local minimum - the specific version presented here is only suitable for local optimization - no "intelligent" greedy agents	- a good quality metaheuristic could be designed by a combination of AC and another metaheuristic	- tune parameters and choices in order to intensify fast allowing for fast convergence - speed up AC to allow for more iterations of above lying metaheuristic

Table 3.5: SWOT matrix on Ant Colony

3.3.5 Genetic Algorithms

The SWOT matrix for the Genetic Algorithm is given by table 3.6. The Genetic Algorithm has a few ways of controlling diversification and intensification. It is widely used for combinatorial problems, but compared to the Evolution Strategies - which is also capable of different divergent and convergent steps - it is hard to see the advantage of the binary approach.

3.3.6 Evolution Strategies

The Evolution Strategies presented in table 3.7 are designed for optimization of continuous functions. It is an evolutionary approach like the Genetic Algorithms with at least as many tools for diversification and intensification.

Genetic Algorithms	*Opportunities* - control diversity by mutation - control diversification/intensification by selection strategy, crossover and mutation - can be hybridized	*Threats* - overlooking minimum solutions close to individual - $(\lambda + \mu)$–strategy will converge population fast
Strengths -convergence by *survival of the fittest* - well described - widely used - mutation and recombination - binary operators allow for fast operations	- adapt search procedure throughout the run of the algorithm -make the iterations fast	- can be combined with LS in order to look for better individuals
Weaknesses - discretization of the solutions - integer approach to optimization of continuous functions - no improvement of individual - does not find local minimum - convergence by *survival of the fittest*	- combine with LS in order to improve final solution	- could be made to converge fast by selection strategy - early interruption would allow for combinations with other algorithms

Table 3.6: SWOT matrix on Genetic Algorithms

Evolution Strategies	*Opportunities* - control diversification/intensification by selection strategy, crossover and mutation - can be hybridized	*Threats* - overlooking minimum solutions close to individual - $(\lambda + \mu)$–strategy will converge population fast
Strengths - developed for continuous functions - works well - convergence by *survival of the fittest* - no discretization - well described	- adapt search procedure	- force converging and early interruption could make ES fast enough to be a sub procedure on a hybrid algorithm
Weaknesses - no improvement of individual - does not find local minimum - convergence by *survival of the fittest*	- combine with LS in order to improve final solution	- could be made to converge fast by selection strategy - early interruption would allow for combinations with other algorithms

Table 3.7: SWOT matrix on Evolution Strategies

3.3.7 Memetic Algorithms

The Evolution Strategies and the Memetic Algorithm are closely related. It is seen from the SWOT matrix in table 3.8 that the possibility of innovation and improvement in the population based approach makes way for even more ways to guide the search. The Memetic Algorithms is clearly the metaheuristic considered in this thesis with the most potential for adapting the divergent and convergent thinking into the search strategy.

3.3.8 Iterated Local Search

The Iterated Local Search procedure presented by table 3.9 is related to the Memetic Algorithm. It does not have advantages over other versions of Memetic Algorithms except it might perform faster iterations. On the other hand several of the opportunities related to divergent and convergent thinking are lost in comparison to the Memetic Algorithm.

3.4 Conclusion

The Memetic Algorithm is selected as the main procedure for implementation and testing. The choice is mainly based on the opportunities presented by the SWOT analysis, but is also affected by the good score in the comparison scheme in table 3.1. The Nelder–Mead simplex is chosen as the direct search heuristic to guarantee a local optimum after each run of the Memetic Algorithm. Furthermore, the simplicity of the Simulated Annealing is seen as an important factor for a possible hybridized approach. In section 5.2 the final version of the Memetic Algorithm is presented together with a hybridized version which uses the Simulated Annealing for parameter optimization.

Memetic Algorithms	*Opportunities*	*Threats*
	- easily applied to continuous functions by using procedures from ES - control diversification/intensification by mutation, meta-mutation and crossover - control diversification/intensification by selection strategy and self-adaptation - could be hybridized - use different kinds of LS - intensify by changing LS method or iterate longer in LS	- $(\lambda + \mu)$–strategy will converge population fast - LS could require much computational time leaving little time for the evolutionary process
Strengths - convergence by *survival of the fittest* - no discretization - powerful hybrid by using LS - finds local optimum - mutation, meta-mutation and crossover	- allows for many possible ways of adjusting the search process - could be utilized as reactive search strategy - focus can shift from evolutionary process to intense neighborhood based LS	- meta-mutation and mutation can be used in order to prevent a population from converging
Weaknesses - convergence by *survival of the fittest* - LS for each individual for both mutation and crossover operations for each iteration is computationally costly	- meta-mutation and mutation can be used in order to prevent a population from converging	- interrupt the LS in most iteration(of evolutionary process) to make the procedure fast

Table 3.8: SWOT matrix on Memetic Algorithms

Iterated Local Search	*Opportunities*	*Threats*
	- use different LS	- hard to tell mutation of a single individual apart from LS
	- intensify by changing LS or iterate longer	- only very similar approaches to converging, i.e. less mutation by auto adaptation on standard deviation and smaller neighborhood
	- control diversification by mutation - could be combined with more extensive LS after final iteration	
Strengths - faster iterations than MA	- few iterations of LS throughout the procedure allows for ultra fast metaheuristic	- implementation of ILS with Nelder–Mead as LS
- computational time used for LS is not that crucial when there is only one individual - finds local minimum - mutation	- the fast version *could* yield good local minima	
Weaknesses - few instruments (parameters) for adjusting the search process		- accept that ILS is quite random on continuous functions and run Nelder–Mead to improve best local minima
- not population based - just an extreme case of MA		

Table 3.9: SWOT matrix on Iterated Local Search

Chapter 4

Divergent and Convergent Thinking as Part of a Creative Approach

4.1 Summary

In this chapter the creative problem solving process based on divergent and convergent thinking is presented in section 4.2. The concept of divergent/convergent steps is taken to the implementation level in section 4.3 by incorporating diversification and intensification in the search strategy of the Memetic Algorithm.

4.2 The Process

The process of finding good solutions to multi-modal continuous functions is designed by a creative approach. The main idea is to use divergent and convergent thinking repeatedly and interactively. The outline of the entire process is given by figure 4.1.

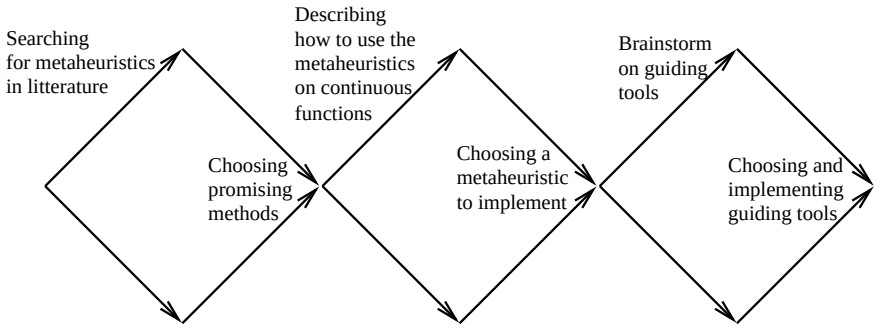


Figure 4.1: The outline of the solution process

This process initiated by the aims formulated in the introduction, i.e. to design a method capable of optimizing continuous functions and even black box problems that are not defined mathematically.

A number of divergent and convergent phases are used as part of a creative problem solving process. According to [26, 27] it is recommendable to start each step of a creative problem solving process by divergent thinking followed by convergent thinking. The brainstorming session is a typical example of divergent thinking. Properties like deferring judgement, combining and quantity is essential for divergent thinking. Structure and systematic approaches are typical for the convergent steps.

The first step approaching the solution in this thesis is divergent in its nature by searching for a large amount of papers and literature in which continuous functions are being solved. Secondly, some metaheuristics based on metaphors from nature are chosen for further investigation in a convergent step. The reason for choosing Descent Methods, Nelder–Mead Simplex, Simulated Annealing, Tabu Search, Ant Colony Optimization, Genetic Algorithms, Evolution Strategies, Memetic Algorithms and Iterated Local Search is that these metaheuristic have either been adapted to solve continuous problems or proved promising in the area of continuous function optimization by the documentation found for combinatorial problems.

The following step contains a detailed description of how to use the metaheuristics on continuous functions. The description based on various papers and literature presents different variations, ideas and possible approaches of utilizing each metaheuristic. Thus, the descriptions can be viewed as a number of parallel divergent steps underlying an overall structure determined by the previous convergent step. Pros and cons are also touched upon in the description in or-

der to make way for a selection process. By convergent thinking the memetic algorithm is chosen for implementation based on a comparison scheme and a modified SWOT analysis. As mentioned in chapter 3 the main objective of the selection of the memetic algorithm is the extensive possibilities of guiding the search strategy in a divergent and convergent way. In that way the divergent/convergent solution process is continued in the test runs of the metaheuristic. In order to decide how the search process of the memetic algorithm should be guided or controlled a brainstorming session is made to come up with a number of suggestion. Only the best suggestions are chosen for implementation in another convergent step. The following section deals with the details of divergent and convergent thinking in the search strategy. Section 4.3 will also deal with the important factors considered when choosing the best guiding tools.

4.3 Divergent and Convergent Thinking as Part of the Search Strategy

In the previous section it is described how a creative solution process is created by repeatedly using divergent and convergent steps in order to approach the solution. In this section, the continuation of the process - into the search strategy of the metaheuristic in order to obtain better solutions - will be described. Basically, Descent Methods can be seen as a way to utilize a convergent strategy. Descent Methods only accept a solution that is closer to minimum. Each descending step is narrowing down the possibilities of a new descending step. At last no descending steps remain. This way the search is converging to a local minimum. Most metaheuristics have some mechanism to escape a local minimum. This can be seen as a divergent step. However, letting the convergent steps of minimization interact with the divergent steps provided by the metaheuristic in a static way might not be a good idea. Consider Simulated Annealing which has a higher probability to escape a local minimum in the start of the search process than in the end. This has proved useful according to both practical tests and thermodynamic theory. The process of making it less likely to escape local minima throughout the search is referred to as intensification. Intensification can be seen as an overall convergent strategy.

Taking this a bit further it could be useful to change strategy throughout the search in a way that both convergent and divergent strategies are utilized repeatedly. An argument for letting this happen is of course that the convergent strategy aids the search for a local minimum, whereas the divergent strategy makes it possible to escape local minima in the search for better local minima. Consider the reason for escaping on the way to a local minimum. The strat-

egy may be divergent in the beginning of a search procedure in order to escape local minima. However, a convergent strategy is needed to reach each of these possible local minima. A useful property of getting close to the local minimum by a convergent strategy before escaping by a divergent strategy is to obtain the actual value of the objective in the local minimum. This way a possible global minimum is not lost because of a too divergent initial strategy. In other words the dilemma of escaping a local minimum before reaching it is that the global minimum might be lost. Likewise, finding a low valued local minimum after utilizing a convergent strategy does not exclude the possibility of switching to another divergent strategy to check for an even lower local minimum. By this reasoning repeatedly shifting between convergent and divergent strategies should be attempted.

Furthermore, the structure of the search space could call for a change of strategy on different levels. Consider a function given by figure 4.2. A search procedure allowing for maximum two consequently ascent moves might be able to find all local minima in the first "valley" given the step size in the middle of the figure. However without diversification the global minimum in the second "valley" cannot be reached. In this case diversification could be obtained by enlarging the step size or increasing the number of allowed ascent moves. In the following the process of going from a divergent search strategy to a convergent search strategy is referred to as intensification whereas, going from convergent to divergent strategy is referred to as diversification. In other words, convergent thinking is represented in the metaheuristic as intensification and divergent thinking appears by diversification.

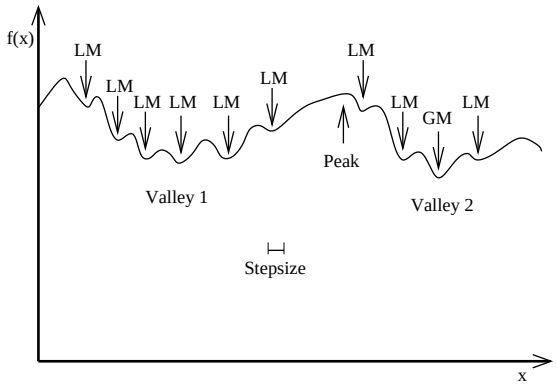


Figure 4.2: Example of search space structure

4.3.1 Controlling Diversification and Intensification

In order to utilize the features of divergent/convergent thinking in the metaheuristic search it is necessary to clarify how intensification and diversification is controlled. Thus, a brainstorming session is made focusing on intensification and diversification in memetic algorithms. The brainstorming session is focused on any tools making diversification or intensification possible, but also on conditions that could be useful in order to trigger either diversification or intensification. The list created by the brainstorm is given by table 4.1.

Some of the tools given by table 4.1 are chosen for implementation. The primary factor for choosing the tools is the estimated effect on either intensification or diversification. However, it is also considered which of the tools can be intuitively explained more clearly. The chosen tools are divided into four groups that are described in the following. The first group contains the threshold ϵ representing the minimal difference in the objective value of individuals in a population. This group represents an overall convergent strategy. The second group contains the local search step size which is a predominately convergent strategy that is broken by diversification when the step size is increased. Third is meta-mutation which is indeed a type of diversification. Fourth is a repeatedly divergent/convergent strategy based on two different search strategies together with an increase and decrease in the number of mutated and reproduced individuals.

4.3.2 Overall Intensification

The threshold ϵ is the minimal difference which is allowed in the objective value of the best individual and the individual ranked $\frac{\mu}{2}$th best, where μ is the population size. Minimal allowed difference means the minimal difference that does not result in a meta-mutation step. Thus, the threshold is closely connected to the diversification described in section 4.3.4. The simple updating of the threshold is given by pseudocode 4.3.1. The decrease of the threshold can be seen as intensification for each generation. The intensification is illustrated by figure 4.3. The use of the threshold in section 4.3.4 means that a smaller threshold would allow the individual that is in the middle of the ranked population to have an objective value closer to that of the best individual. If the two objective values where too close it would result in meta-mutation (diversification). Therefore, the idea of decreasing the limit is actually intensification. On the other hand, as a result of other convergent properties in the metaheuristic the population is more likely to contain more individuals with the same objective value in the end of the search than in the beginning of the search. From that point of view the decreasing threshold can be viewed more like an adaptation

- dynamic population size (μ)
- meta-mutation
- random restart
- maximum number of identical individuals
- dynamic mutation size $\lambda_{\text{mutation}}$
- dynamic reproduction size $\lambda_{\text{reproduction}}$
- no improvement of best individual
- number of iterations without improvement of best individual
- $(\lambda + \mu)$–strategy (intensification)
- (λ, μ)–strategy (diversification)
- number of local search iterations
- dynamic local search step size
- different strategies for the mutation operator
- maximum standard deviation $\sigma_{\max}$ in the mutation
- minimal improvement of best individual
- self adaptation of the standard deviation σ
- minimal distance between individuals in the search space
- change between diversification/intensification according to fixed interval
- minimal difference in the objective value of individuals in one population (ϵ)

Table 4.1: Brainstorm on diversification/intensification

than an intensification.

Pseudocode 4.3.1 Controlling ϵ

1: $\epsilon_t \leftarrow \epsilon_{t-1} \cdot \text{dec}_\epsilon$, where $\text{dec}_\epsilon \in]0; 1[$

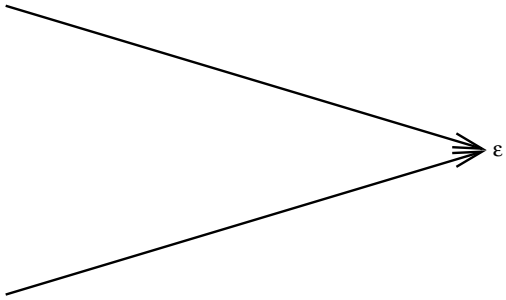


Figure 4.3: The overall convergence by decreasing threshold ϵ

4.3.3 Predominately Intensification

The local search step size is decreased in every generation (iteration) in order to intensify the search if the best individual is improving. However, in case the best individual in a generation has not got a lower objective value than the best individual in the generation before the intensification is broken by a divergent step. The intensification and diversification is given by the lines in pseudocode 4.3.2. Given that the best individual improves, the step size is decreased in each generation, line 1–2. In case the best individual is not improving the step size is increased instead, line 3–4. The intensification which is from time to time broken by diversification is illustrated graphically by figure 4.4.

Pseudocode 4.3.2 Controlling StepSize

1: **if** $f(\mathbf{x}^{(\text{best},t)}) < f(\mathbf{x}^{(\text{best},t-1)})$ **then**
2: $\quad \text{StepSize}_t \leftarrow \text{StepSize}_{t-1} \cdot \text{dec}, \text{ where dec} \in]0;1[$
3: **else**
4: $\quad \text{StepSize}_t \leftarrow \text{StepSize}_{t-1} \cdot \text{inc}, \text{ where inc} > 1$
5: **end if**

4.3.4 Diversification by Meta-mutation

The meta-mutation is a somewhat drastically divergent step. It mutates all individuals except the best. The meta-mutation procedure is given by pseudocode 4.3.3. Meta-mutation is carried out in case at least one of the following two conditions given by line 2 holds. One, the difference between the objective value of the best individual and the individual on the $\frac{\mu}{2}$th place of the ranked

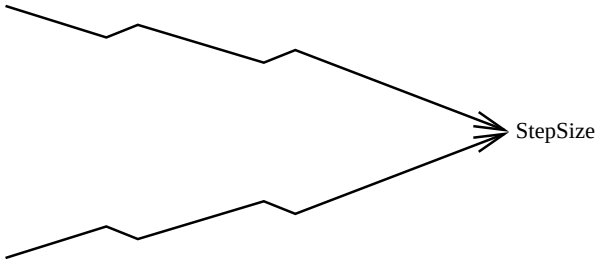


Figure 4.4: The predominant convergence of the local search stepsize

population is below a certain threshold ϵ[1]. Two, the diversity of the population is below a specified minimum diversity. For this purpose the diversity of the population is calculated as the sum of differences over all variables between the best solution and all of the remaining 66 % of the best solutions. Meta-mutation is carried out by mutating each gene of all individuals except the best, line 3–4. The actual mutation of the gene is done by adding a normally distributed number with mean zero and standard deviation $k \cdot \sigma_i^{(l,t)}$, where k_{scale} is a constant determining how diverse the meta-mutation should be and $\sigma_i^{(l,t)}$ is a genetic property initialized by the memetic algorithm.

Pseudocode 4.3.3 Procedure Meta-mutation

```
1: Population_t ← Ranked(Population_t)
2: if f(x^(best,t)) + ε ≥ f(x^(μ/2,t)) or Diversity(Population_t) < minDiversity then
3:    for All x^(l,t) ∈ Population_t \ {x^(best,t)} do
4:       for i = 1 to n do
5:          x_i^(l,t) ← x_i^(l,t) + k_scale · N(0, σ_i^(l,t))
6:       end for
7:       x^(l,t) ← LocalSearch(x^(l,t), σ^(l,t))
8:    end for
9: end if
```

The diversification caused by meta-mutation after the population is converged is illustrated by figure 4.5. It is seen that the population slowly converges over several steps to a certain extinct. Meta-mutation is then triggered causing the population to diverge rapidly in one step. As implied by figure 4.5, the diversity of the population never becomes as large as the initial diversity. A further illustration of the diversity is given by figure 4.6, which is based on the output from optimizing the Branin[2] function given in appendix 9.2. The figure shows

[1] ϵ is decreased as described earlier in section 4.3

[2] The reason for using the Branin function for illustrative purposes is that it only has two

the converged population as blue circles each representing an individual. The green crosses represent individuals in the population after meta-mutation and one step of local search. Although a little hard to tell, it can be seen from the illustration that exactly one circle contains a cross. That individual is the best individual who is not mutated. Figure 4.6 also illustrates the fact that meta-

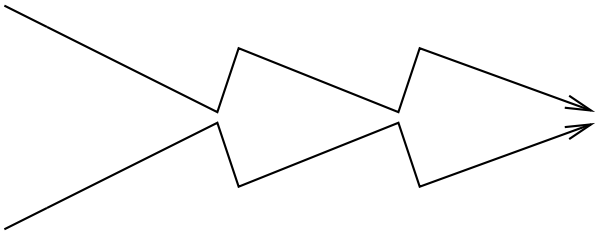


Figure 4.5: The divergence caused by meta-mutation

mutation does not result in a population as diverse as the initial population. In the example all individuals lie within the range $(x_1, x_2) \in ([2.5; 4.0], [1.8; 2.8])$ even though the search space of the Branin function is much larger. The main reason for this is obviously that meta-mutation should make some connection to the converged population instead of just initialize another random population. Another reason is that meta-mutation is always followed by a local search step as well as any other mutation and reproduction procedure is followed by local search.

A Further Remark to the Diversity One might argue that the meta-mutation should cause the individuals to be spread around the entire search space. The argument not to do that is that meta-mutation is *not* a random restart. One could argue further that another diversification procedure as the random restart could be utilized parallel to the meta-mutation for even harder diversification. The argument not to let the Memetic Algorithm or any other metaheuristic restart and initialize all over again is that restart can be seen as cheating when considering a test setup as follows. For testing purposes the algorithm would indeed run more than once, but only taking the best out of a number of solutions does not give a very good comparison to other heuristics. Throwing away the average would be throwing away information. This might yield the result that two algorithms seem equally good although they might produce very different averages in the long run. One might ask what testing has to do with the internal procedures of the algorithm. If there is time to restart the algorithm within each run this could be done, but then the maximum

variables which suits a two dimensional illustration well

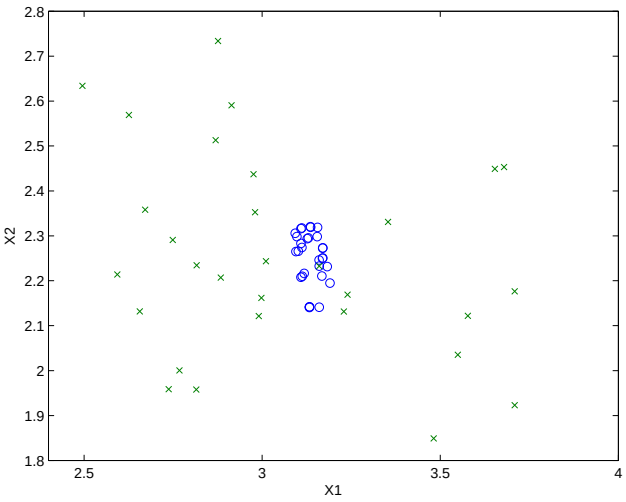


Figure 4.6: The population before and after meta-mutation

running time might as well be cut down. After all, there is always the possibility of running the algorithm a number of times in the test phase. For real-life problems one would surely take the best solution, which might imply that the less good solutions are unnecessary. However, this does not change the fact that for testing purposes it is easier to compare algorithms by the average and the deviation than by the best objective solely.

4.3.5 Dynamic Number of Mutations and Reproductions

The final group of tools to control the search procedure is perhaps the most important as it calls for changes in the selection strategy influencing the search to a large extinct. The selection strategy used for diversification is the (λ, μ)–strategy whereas, the $(\lambda + \mu)$–strategy is used for intensification. The strategies are both described in section 2.11.3. The (λ, μ)–strategy selects the next generation amongst the offspring and mutated individuals only. The $(\lambda + \mu)$–strategy is clearly more convergent as it allows parents to compete with the offspring. Another tool for intensification is to adjust the number of offspring produced from reproduction and mutation. It might be trivial to see that a larger number of mutations is a divergent change whereas, a larger number of reproductions between parents is a less clear convergent step. Reproduction does after all in-

troduce diversity into a population of parents. On the other hand, the offspring is an exact mixture of the two parents. However, different from both parents, the offspring does lie somewhere in between the parents. It is decided that the divergent steps should increase the number of mutations $\lambda_{\text{mutation}}$ on the expense of the number of reproductions $\lambda_{\text{reproduction}}$. The convergent steps should make the opposite changes.

The divergent and convergent steps are given by pseudocode 4.3.4. For every t_{interval} generation the procedure shifts between t_{interval} consecutive generations of diversification or intensification, line 1 pseudocode 4.3.4. The diversification/intensification process based on the fixed interval t_{interval} is illustrated by figure 4.7. In case of diversification, the mutation rate is increased and the reproduction rate is decreased by one, line 2–3. Furthermore the temporary population used for selection includes only the offspring from reproduction together with the mutated individuals, line 7. On the other hand, the reproduction rate is increased whereas, the mutation rate is decreased by one when intensification appears, line 10–11. As for the original procedure described in section 2.12.2, the selection is based on both parents, offspring and mutated individuals, line 15. Note that the mutation rate as well as the reproduction rate is in fact bounded by a lower value in order not to make λ_{mutate} and $\lambda_{\text{reproduce}}$ negative.

Pseudocode 4.3.4 Procedure DivInt

1: **if** $t \mod (2 \cdot t_{\text{inteval}}) \geq t_{\text{interval}}$ **then**
2: $\lambda_{\text{mutate}} \leftarrow \lambda_{\text{mutate}} + 1$
3: $\lambda_{\text{reproduce}} \leftarrow \lambda_{\text{reproduce}} - 1$
4: $\text{Offspring}_t \leftarrow \text{Reproduction}(\text{Population}_t)$
5: $\text{Population}_{\text{temp}} \leftarrow \text{Population}_t \cup \text{Offspring}_t$
6: $\text{Population}_{\text{mutated},t} \leftarrow \text{Mutation}(\text{Population}_{\text{temp}})$
7: $\text{Population}_{\text{temp}} \leftarrow \text{Offspring}_t \cup \text{Population}_{\text{mutated},t}$
8: $\text{Population}_{t+1} \leftarrow \text{Selection}(\text{Population}_{\text{temp}})$
9: **else**
10: $\lambda_{\text{reproduce}} \leftarrow \lambda_{\text{reproduce}} + 1$
11: $\lambda_{\text{mutate}} \leftarrow \lambda_{\text{mutate}} - 1$
12: $\text{Offspring}_t \leftarrow \text{Reproduction}(\text{Population}_t)$
13: $\text{Population}_{\text{temp}} \leftarrow \text{Population}_t \cup \text{Offspring}_t$
14: $\text{Population}_{\text{mutated},t} \leftarrow \text{Mutation}(\text{Population}_{\text{temp}})$
15: $\text{Population}_{\text{temp}} \leftarrow \text{Population}_t \cup \text{Offspring}_t \cup \text{Population}_{\text{mutated},t}$
16: $\text{Population}_{t+1} \leftarrow \text{Selection}(\text{Population}_{\text{temp}})$
17: **end if**

To illustrate a part of the progress of the search procedure another output based on the Branin function is presented in figure 4.8. The blue line illustrates the objective value of the overall best individual as the generations evolve. The current best objective is given by the red line. Black blocks on the horizontal axis

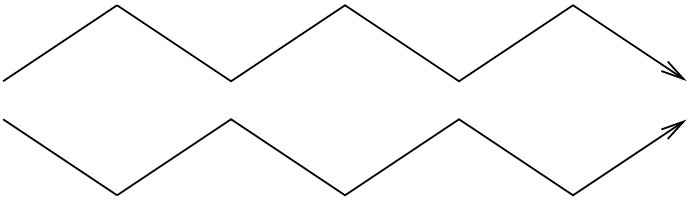


Figure 4.7: The shifting between convergent and divergent selection strategy

show the diversification interval whereas the generations in between the black blocks utilize intensification. In this particular example the interval t_{interval} is set to nine generations. It is seen that the objective value of the current best individual is able to either increase, decrease or stay the same. However, the current best can only increase its objective value in the divergent steps above the black blocks. The reason for this is the ability of survival of the fit parents when subject to intensification, and the inability of survival of a parent in case of diversification no matter fitness.

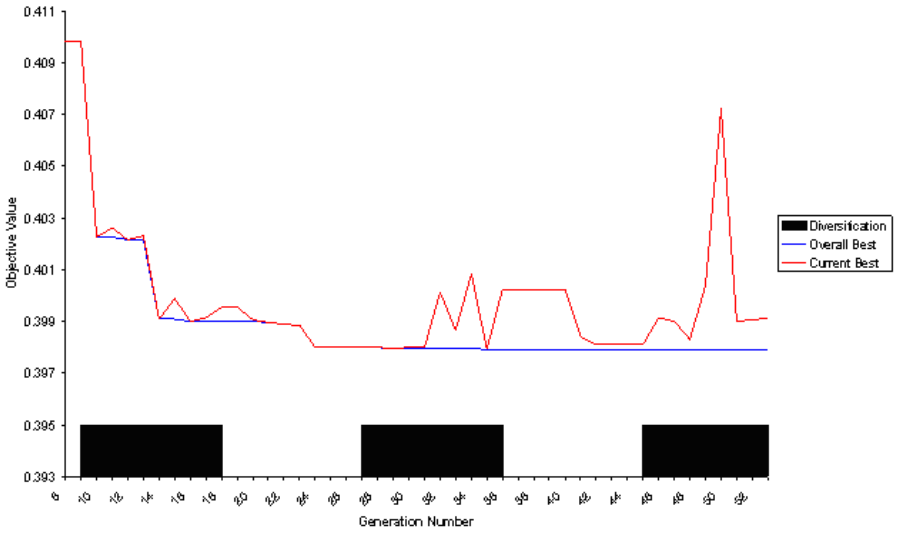


Figure 4.8: The progress of the search procedure

Chapter 5

The Final Algorithm and Implementation

5.1 Summary

In this chapter the procedure of the final metaheuristic is outlined. Furthermore, a summary of the object oriented JAVA-implementation is given with reference to the source code in appendix .9.

5.2 The Final Memetic Algorithm

The procedure of the final Memetic Algorithm is a product of the general procedure given by section 2.12.2 and the divergent and convergent strategies developed in section 4.3. Although the procedure has close similarities to the general algorithm, crucial differences exist. Thus, the final procedure will be described in detail with special focus on the various parameters affecting the search strategy.

5.2.1 Parameters

All the parameters are given by the first column of table 5.1. In the second column a short description is made. Some parameters are adapted to the search space either in general or for each dimension of the solution. The third column denote the parameters subject to adaptation by a $\surd$. The maximum running time will be highlighted as a parameter in the pseudocode, because it is an input to the procedure defined before each run. However, it is not contained in the table since, it is not a parameter that calls for adjustment in any way. The maximum running time is set to 5000 ms for all tests, except when utilizing external simulations (section 6.3).

Parameter	Description	Adaptation
μ	The population size	
$\lambda_{\text{reproduce}}$	The initial number of offspring	
λ_{mutate}	The initial number of mutated individuals	
t_{interval}	Generations between diversification/intensification	
t_{init}	Initial number of diversification generations	
σ_{max}	The maximum standard deviation for mutation procedure	$\surd$
StepSize	The initial step size for the local search procedure	$\surd$
dec	The factor decreasing the local search step size	dec $\in]0;1[$
inc	The factor increasing the local search step size	inc > 1
ϵ	The threshold value which can trigger the meta-mutation	$\surd$
dec_ϵ	The factor decreasing epsilon at each generation	$\text{dec}_\epsilon \in]0;1[$
minDiversity	The minimum diversity of the individuals of the population	$\surd$
k_{scale}	The scale of the population's diversity after meta-mutation	$\surd$
LS_{steps}	The number of local search steps performed	
$\text{LS}_{\text{max-try}}$	The maximum number of candidates in a local search step	
$\theta_{\text{diversity}}$	Best part of the population considered	$\theta_{\text{diversity}} \in [1;\mu]$

Table 5.1: Parameters of the Memetic Algorithm

5.2.2 Algorithm

The procedure of the Memetic Algorithm is described below in detail, assisted by a pseudocode. The pseudocode uses a color coding where, the parts identical to the general algorithm from section 2.12.2 is given in black. The new parts related to divergent and convergent thinking introduced in section 4.3 are given in blue. Finally, are the parameters in red and the parts not mentioned earlier are typed in green.

The main part of the final Memetic Algorithm is given by pseudocode 5.2.2. The first twelve lines are similar to that of the original procedure. Each of the μ individual of the population are initialized in all dimensions, line 3–9 pseudocode 5.2.2. The individuals are improved by local search before being added to the population, line 10–11. The stopping criterion of the final procedure is specifically defined to be a maximum running time $\max_{\text{runtime}}$ given by the difference between the start time and the current time, line 13.

At an interval of t_{interval} the algorithm switches between diversification and intensification. However, in case of the first t_{init} generation intensification will be used, line 14. The intensification step increases the number of mutations by one on the expense of the number of reproductions, line 15–16. Furthermore, selection for the next generation is only based on the offspring from reproduction and the mutated individuals, line 20–21. In case of diversification, the number of offspring is increased and the number of mutations is decreased by one, line 23–24. The next population is chosen amongst both parents, offspring and mutated individuals, line 28–29.

5.2.2.1 Meta-mutation

Meta-mutation is performed under either one of two conditions. First, in case the objective value of the best individual is not lower than the objective value ranked $\frac{\mu}{2}$ from the best, within a certain threshold ϵ. Second, in case the diversity of the ranked population is less than a specified value minDiversity, line 32. The diversity is defined by pseudocode 5.2.1 as the sum over all n dimensions of the absolute value of the difference between the solution of the best individual and the solution of the first $\theta_{\text{diversity}}$ of the ranked individuals. As seen from pseudocode 5.2.1 the definition of diversity does not consider the objective values, only the distance in the search space between the best individual and the surrounding individuals.

Pseudocode 5.2.1 Procedure Diversity

1: Diversity$\leftarrow \sum_{k=1}^{\theta_{\text{diversity}}} \sum_{i=1}^{n} |\mathbf{x}_i^{\text{best},t} - \mathbf{x}_i^{k,t}|$
2: **return** Diversity

The meta-mutation updates all individuals except the best, line 33. Each gene is modified by adding a normally distributed random number with mean zero and a scaled version of the standard deviation given by the genes of the individual, line 34–35. Each mutated individual is eventually improved by local search, line 37.

The step size is either decreased of increased depending on the improvement of the best individual in the current generation, line 40–44. Furthermore the threshold is decreased as part of an overall convergent strategy, line 45. Finally, the $n + 1$ best individuals of the ranked population are utilized to find a local minimum by the Nelder–Mead simplex procedure, line 48–53. The overall best solution is returned in line 54. The Nelder–Mead procedure is identical to the original described in section 2.5. Even the parameters are set to the common universal.

5.2.2.2 Local Search

The local search procedure used after each mutation, reproduction and meta-mutation step is given by pseudocode 5.2.3. The local search performs a limited number of steps $\mathrm{LS}_{\mathrm{steps}}$ before terminating, line 1 pseudocode 5.2.3. The procedure keeps modifying the solution as long as a candidate solution has a worse objective value than the current solution and a maximum number of trials is not reached, line 3. The candidate is created by adding a random step to each dimension of the current solution, line 4–6. The standard deviation is duplicated from the current solution, line 7. In case a better candidate is found the current solution is updated.

Pseudocode 5.2.2 Procedure MA

```
1: t ← 1
2: Population_t ← ∅
3: for k = 1 to μ do
4:    for i = 1 to n do
5:       draw random χ from uniform interval [0; 1]
6:       x_i^(k,t) ← u_i + χ(v_i − u_i)
7:       draw random φ from uniform interval [0; 1]
8:       σ_i^(k,t) ← φ · σ_max
9:    end for
10:   x^(k,t) ← LocalSearch(x^(k,t), σ^(k,t))
11:   Population_t ← Population_t ∪ {(x^(k,t), σ^(k,t))}
12: end for
13: while Time(0) - Time(t) < max_runtime do
14:   if t mod (2 · t_interval) ≥ t_interval or t < t_init then
15:      λ_mutate ← λ_mutate + 1
16:      λ_reproduce ← λ_reproduce − 1
17:      Offspring_t ← Reproduction(Population_t)
18:      Population_temp ← Population_t ∪ Offspring_t
19:      Population_mutated,t ← Mutation(Population_temp)
20:      Population_temp ← Offspring_t ∪ Population_mutated,t
21:      Population_t+1 ← Selection(Population_temp)
22:   else
23:      λ_reproduce ← λ_reproduce + 1
24:      λ_mutate ← λ_mutate − 1
25:      Offspring_t ← Reproduction(Population_t)
26:      Population_temp ← Population_t ∪ Offspring_t
27:      Population_mutated,t ← Mutation(Population_temp)
28:      Population_temp ← Population_t ∪ Offspring_t ∪ Population_mutated,t
29:      Population_t+1 ← Selection(Population_temp)
30:   end if
31:   Population_t ← Ranked(Population_t)
32:   if f(x^(best,t)) + ε ≥ f(x^(μ/2,t)) or Diversity(Population_t) < minDiversity then
33:      for All x^(l,t) ∈ Population_t \ {x^(best,t)} do
34:         for i = 1 to n do
35:            x_i^(l,t) ← x_i^(l,t) + k_scale · N(0, σ_i^(l,t))
36:         end for
37:         x^(l,t) ← LocalSearch(x^(l,t), σ^(l,t))
38:      end for
39:   end if
40:   if f(x^(best,t+1)) < f(x^(best,t)) then
41:      StepSize_t+1 ← StepSize_t ·dec, where dec ∈]0; 1[
42:   else
43:      StepSize_t+1 ← StepSize_t ·inc, where inc > 1
44:   end if
45:   ε_t ← ε_t−1 · dec_ε, where dec_ε ∈]0; 1[
46:   t ← t + 1
47: end while
48: Population_t ← Ranked(Population_t)
49: Population_NM ← ∅
50: for k = 1 to n + 1 do
51:    Population_NM ← Population_NM ∪ {x^k,t}
52: end for
53: x^best ← Nelder–Mead(Population_NM)
54: return x^best
```

Pseudocode 5.2.3 Procedure LocalSearch

```
1:  for Step = 0 to LS_steps do
2:     count← 0
3:     while f(x^candidate) ≥ f(x^current) and count < LS_max−try do
4:        for i = 1 to n do
5:           draw random χ from uniform interval [−0.5; 0.5]
6:           x_i^candidate ← x_i^current + stepSize · χ
7:           σ_i^candidate ← σ_i^current
8:        end for
9:     end while
10:    if f(x^candidate) < f(x^current) then
11:       x^current ← x^candidate
12:    end if
13: end for
14: return x^current
```

5.2.2.3 Reproduction

The reproduction procedure used in the Memetic Algorithm is given by pseudocode 5.2.4. This sub procedure is identical to that of the original Memetic Algorithm already described. The individual for reproduction are chosen at random from the population, line 3 pseudocode 5.2.4. Each gene is taken from one of the parents by equal probability, line 5–9. Each offspring is finally improved by a local search, line 12.

Pseudocode 5.2.4 Procedure Reproduction

```
1:  Offspring_t ← ∅
2:  for l = 1 to λ_reproduce do
3:     draw two integers k and m at random from interval {1, 2, ..., μ}
4:     for i to n do
5:        draw random χ from uniform interval [0; 1[
6:        if χ < 0.5 then
7:           γ_i^l ← ((x_i^(k,t))(σ_i^(k,t)))
8:        else
9:           γ_i^l ← ((x_i^(m,t))(σ_i^(m,t)))
10:       end if
11:    end for
12:    γ^l ←LocalSearch(γ^l)
13:    Offspring_t ← Offspring_t ∪ {γ^l}
14: end for
```

5.2.2.4 Mutation

The mutation procedure is applied to λ_{mutate} random individuals of the temporary population, line 1–2 pseudocode 5.2.5. Each gene is mutated by adding a normally distributed random number with mean zero and a standard deviation given by the gene of the individual, line 3–4. The mutated individuals are also subject to a local search improvement, line 6–7.

Pseudocode 5.2.5 Procedure Mutation

1: **for** $l = 1$ to λ_{mutate} **do**
2: draw an integer m at random from interval $\{1, 2, \ldots, \mu + \lambda_{\text{reproduce}}\}$
3: **for** $i = 1$ to n **do**
4: $x_i^{(l,t)} \leftarrow x_i^{(m,t)} + N(0, \sigma_i^{(m,t)})$
5: **end for**
6: $\beta^l \leftarrow \text{LocalSearch}\big((\mathbf{x}^{(l,t)}), (\sigma^{(m,t)})\big)$
7: $\text{Population}_{\text{mutated},t} \leftarrow \text{Population}_{\text{mutated},t} \cup (\beta^l)$
8: **end for**

5.2.2.5 Selection

The selection procedure given by pseudocode 5.2.6 is also identical to that of the original algorithm described in section 2.12.2. The new population is simply chosen as the μ best individuals, i.e. the individuals with the lowest objective values in the temporary population. The different selection strategies relies on which temporary population is subject to selection. In this way the selection strategy is chosen in the main procedure described in section 5.2.2

Pseudocode 5.2.6 Procedure Selection

1: $\text{Population}_{t+1} \leftarrow \varnothing$
2: Let $\alpha \in \text{Population}_{temp}$
3: **for** $k = 1$ to μ **do**
4: $(\mathbf{x}^{(k,t)}, \sigma^{(k,t)}) \leftarrow \alpha | \{\min f(\alpha), \forall \alpha \in \text{Population}_{temp}\}$
5: $\text{Population}_{temp} \leftarrow \text{Population}_{temp} \backslash \{(\mathbf{x}^{(k.t)}, \sigma^{(k,t)})\}$
6: $\text{Population}_{t+1} \leftarrow \text{Population}_{t+1} \cup \{(\mathbf{x}^{(k,t)}, \sigma^{(k,t)})\}$
7: **end for**

5.2.3 Automatic Parameter Finding

As seen from table 5.1 the Memetic Algorithm ends up with quite a large number of parameters. Some parameter settings might produce good solutions whereas, others might lead to poor results. A parameter tuning is made as the first part of the results in section 6.1. However, an alternative solution to this problem allowing for discovery of good solutions is also developed. Another metaheuristic is used to keep track of the results of the Memetic Algorithm subject to changes in the parameter values. The metaheuristic on top should be allowed to keep good results, but also to escape non-improving results as an "innovative" step. Thus, the metaheuristic should be capable of escaping local minima. On the other hand, it is desirable to keep it simple with regard to the number of function evaluations and the number of parameters in order to make it easy to decide the parameter values of the new metaheuristic running on top. For this purpose the Simulated Annealing is used. The procedure is identical to that of section 2.4.2. Figure 5.1 illustrates the framework of the metaheuristics and the original problem given by a continuous function. Each function evaluation of the Simulated Annealing running on top of the Memetic Algorithm returns the average objective value of a specified number of runs of the Memetic Algorithm. A solution considered by the Simulated Annealing consists of a number of parameter values for the Memetic Algorithm. The parameters and the boundaries defining the search space are given by table 5.2. The choice of

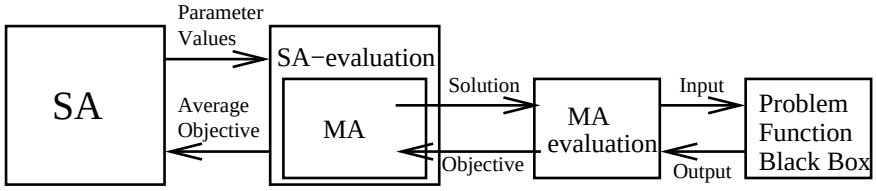


Figure 5.1: The framework of metaheuristics and the problem

boundaries and fixed parameters reflect the choice made for parameter tuning. In section 6.1 the reason for setting some parameters to a fixed value is given. To allow all parameters to move around the continuous search space the value of $\mu, \lambda_{\text{reproduce}}, \lambda_{\text{mutate}}$ and t_{interval} are rounded at each function evaluation.

The combined version of a Simulated Annealing running on top of the Memetic Algorithm and the Nelder–Mead simplex should not only be seen as an attempt to find optimal parameter values for the normal algorithm in section 5.2.2. The intension is rather to find optimal values to the original continuous function by other means. The parameter values passed from the Simulated Annealing to the Memetic Algorithm are not interesting as such. However, the values

Parameter	Search space
μ	[15;30]
$\lambda_{\text{reproduce}}$	[15;55]
λ_{mutate}	[15;55]
t_{interval}	[5;9]
t_{init}	7
$\sigma_{\max}$	[0.1;0.3]
StepSize	0.09
dec	[0.8;0.9]
inc	[1.03;1.09]
ϵ	[0.01;0.02]
dec_ϵ	[0.4;0.7]
minDiversity	0.1
k_{scale}	6.0
LS_{steps}	1
$\text{LS}_{\text{max-try}}$	50
$\theta_{\text{diversity}}$	0.66

Table 5.2: Parameters being variables

lead to the discovery of the best solution with the lowest average objective value. An argument that the parameter values themselves are not interesting is the crucial difference between the nature of a normal optimization task and the task faced by the Simulated Annealing. The function evaluation of the Simulated Annealing cannot even be seen as a black box function. Parameter values leading to a specific average value when evaluated are not guaranteed to lead to that specific average value in other cases. Using the metaphor of a black box; the input to the box does not necessarily yield the same output always. However, considering the original problem given by a continuous function this does not matter. One is only interested in the best solution and the objective value of that solution.

5.3 Implementation

The implementation of the program is done in the JAVA programming language and the source code can be found in appendix .9. The implementation consists of the following five files/objects:

1. The MainClass.java is the main file. The program is started by this file

which also includes the loops for parameter tuning and testing. MainClass.java is included in appendix .10.

2. In appendix .14 the SA.java file contains the Simulated Annealing procedure used for improving the parameters and the output from the Memetic Algorithm.

3. The MA.java given by appendix .11 contains the core of the program. All sub procedures of the Memetic Algorithm and the Nelder–Mead simplex are implemented here.

4. The DataObject.java holds the data to be processed by the program. This file takes care of reading problem instances from some problems e.g. the function matrices of the Hartmann functions. Furthermore, it reads the output data from simulations of external programs which are considered as black boxes to the optimization algorithm. Finally, this file contains a number of objective functions used for solution evaluation of each problem.

5. The LogFile.java handles the logging of the results achieved by the optimization procedure. This file is also used for writing the input data for external simulations, i.e. output from the optimization procedure is input to the external program and vice versa.

In MA.java a population of the Memetic Algorithm is represented by a three dimensional object in the program. A vector of variable size holds a number of individuals. Each individual is represented by three arrays containing the genes representing the solution, the genes representing the standard deviation and the objective value if the individual is already evaluated by the DataObject.java. The reason for attaching the objective value to the memory of each individual is the costly process of evaluating the solution. This is especially important due to the long running time of the external simulation program.

Chapter 6

Results

This chapter presents the different results. The first section describes the parameter tuning of the Memetic Algorithm on a subset of functions to be optimized. The mathematically defined test functions are given in appendix 9.2 and the tests based on the Memetic Algorithm are compared to that of the Directed Tabu Search (DTS) by [14]. The results from the combined Simulated Annealing and Memetic Algorithms are also described although not directly comparable.

In section 6.3 a problem considering an external simulation model of groundwater flow is described. Finding parameters to the external simulation is an example of a black box optimization problem. The results of applying the Memetic Algorithm and the creative search strategy to this problem are compared to manual calibration in section 6.3.

6.1 Parameter Tuning

In order to test the Memetic Algorithm the parameters should be tuned. This is done by testing different parameter combinations on a subset of the mathematically defined non-linear continuous functions. For representative testing another subset is used. However, the functions considered can be quite different in nature and the parameters can therefore be quite far from optimal for some

problems. This is the trade off when an algorithm and parameter setting capable of solving various problems is wanted. Although the external simulation model in section 6.3 is not even a mathematically defined function it utilizes the same parameter settings with only a couple of necessary changes[1].

The different values used for parameter tuning are given by table 6.1. Due to the number of parameters the number of different combinations have to be restricted. Thus, the parameters assumed to have the least effect on the search are fixed to a reasonable value. The total number of possible combinations is 2700. The parameter values are all within the intervals used as the search space for the simulated annealing in section 5.2.3. The μ values are not allowed to exceed 30 because the selection procedure becomes less significant the larger μ gets. Furthermore the minimum number of reproduced and mutated individuals should at least be as high as the maximum number of selected individuals for the (λ, μ)–strategy. The interval t_{interval} between diversification and intensification is minimum five, since it is considered difficult to trace the effect of a lower number of steps. The reason not to let the interval be too large is that a number of diversification/intensification steps should be allowed before termination, even if iterations are time consuming. The number of initial steps t_{init} of diversification is fixed to seven. The idea is to guarantee a divergent search in the start of the procedure. The number could be higher but as seen from section 4 the population can then converge in other ways.

The maximum standard deviation $\sigma_{\max}$ is restricted to three possible values relative to the search space. The step size is increased and decreased as described earlier. Therefore, the initial value is less important and is fixed to only one possible value. The relative decrease and increase of the step size is likewise fixed. The threshold ϵ is allowed to take one of three possible values. No matter which initial threshold is used the relative decrease is fixed to 0.65 in order for ϵ to approach zero relatively fast. Fixing the minimum diversity to 0.1 has shown by trials to produce meta-mutation steps relatively often throughout the search procedure for different problems. An example of the improvement and steps of the search procedure can be seen from appendix .14. Fixing the scale k_{scale} of the meta-mutation to six is done based on plots similar to figure 4.5 on page 63, illustrating the diversity of the population before and after meta-mutation.

The number of local search steps LS_{steps} is set to one. The main reason for allowing only one step is that local search is utilized at every iteration in any other operation than the selection procedure. This means that the time consumption of the overall procedure is highly dependant on the time consumption of the local search. On the other hand, it is important to have at least one local search step because it hybridizes the evolutionary algorithm as described earlier. The

[1] The population sizes have to be decreased in order to obtain results in reasonable time

maximum number of trials in the local search procedure $LS_{\mathrm{max-try}}$ is fixed to 50. This value should not have significant influence on the search procedure in practice. It is there to make sure the local search procedure will terminate in case it is at a local minimum. However, for time consuming function evaluations it is desirable to decrease this value not to waste time. In fact $LS_{\mathrm{max-try}}$ is fixed to ten for the optimization of the external ground water simulation. The idea is that the possibility of finding an improved solution past the tenth trial is low if no improved solution could be found within ten trials.

The last parameter $\theta_{\mathrm{diversity}}$ representing the part of the population which is considered in the diversity function is fixed to 66 % . The thought behind this is that the function obviously has to take a number of individuals into account. To measure if the population is completely converged all individuals would have to be taken into account. On the other hand, if the best part of the population is converged there might not be much innovation from that part anymore. Thus, it is desirable to introduce metamutation when only the best has converged. In this case $\theta_{\mathrm{diversity}}$ is set to the best 66 % of the individuals.

Parameter	Setting
μ	15, 20, 25, 30
$\lambda_{\mathrm{reproduce}}$	15,25,35,45,55
$\lambda_{\mathrm{mutate}}$	15,25,35,45,55
t_{interval}	5,7,9
t_{init}	7
σ_{max}	0.1,0.2,0.3
StepSize	0.09
dec	0.9
inc	1.03
ϵ	0.01, 0.015, 0.02]
dec_{ϵ}	0.65
minDiversity	0.1
k_{scale}	6.0
$\mathrm{LS}_{\mathrm{steps}}$	1
$\mathrm{LS}_{\mathrm{max-try}}$	50
$\theta_{\mathrm{diversity}}$	0.66

Table 6.1: Parameter tuning values

The parameter tuning is based on the *Branin* function and the $Shekel_{4,7}$ function described in appendix 9.2. Each of the parameter combinations is tested by five runs of each of the two functions. Each run is stopped after five seconds. A *success rate* is calculated by dividing the number of *successful* runs by the

total number of five runs, i.e. $\frac{\text{runs}_{\text{success}}}{5}$. A successful trial run is defined by the criterion in equation (6.1), as a run that has a final objective $f(\mathbf{x})$ value within some range of the objective value of the optimal solution $f(\mathbf{x}^*)$. The optimal objective values for each function are given in appendix 9.2. The success criterion in equation (6.1) is identical to that of [14].

$$|f(\mathbf{x}^*) - f(\mathbf{x})| < 10^{-4} \cdot |f(\mathbf{x}^*)| + 10^{-6} \tag{6.1}$$

An *average error* is defined by equation (6.2) as the average relative gap between the objective value of the optimal solution and the objective value of the final solution in each successful run.

$$\text{Avg.Err.} = \frac{\sum_{i=1}^{5} \left| \frac{f(\mathbf{x}^*) - f(\mathbf{x}_i)}{f(\mathbf{x}^*)} \right|}{5 \cdot \text{Suc.rate}}, \quad \text{where equation 6.1 holds for } \mathbf{x}_i \tag{6.2}$$

Finally a *standard deviation* δ is defined by equation (6.3) based on the successful runs.

$$\delta = \sqrt{\frac{\sum_{i=1}^{5} \left(\text{Avg.Err.} - \left| \frac{f(\mathbf{x}^*) - f(\mathbf{x}_i)}{f(\mathbf{x}^*)} \right| \right)^2}{5 \cdot \text{Suc.rate}}}, \quad \text{where equation 6.1 holds for } \mathbf{x}_i \tag{6.3}$$

The results of the parameter tuning are displayed by table 6.2. The table contains the 35 best parameter combinations when ordering the results primarily by the success rate, then by the average error and finally by the standard deviation. The success rate, average error and standard deviation are all calculated as averages of the tests based on the Branin and the $\text{Shekel}_{4,7}$ functions. Similar tables of each test individually are given in appendix 1 and 2. The parameter settings are given by the first six columns of table 6.2. The parameter settings in the first line are chosen for the test runs in section 6.2. The best parameter combination with a success rate of 0.9 has a lower average error than the best parameter combination with a success rate of 1.0. However, the success rate is prioritized higher than the average error. Thus, the parameter values for the test runs are fixed to:

$$\mu = 30, \quad \lambda_{\text{reproduce}} = 15, \quad \lambda_{\text{mutate}} = 55, \quad \sigma_{\max} = 0.2, \quad \epsilon = 0.02 \text{ and } t_{\text{interval}} = 9 \tag{6.4}$$

6.2 Mathematically Defined Functions

The Memetic Algorithm with a divergent/convergent search strategy is tested on five continuous functions defined by appendix 9.2. Each test consists of 100

μ	λ_{rep}	λ_{mut}	$\sigma_{\max}$	ϵ	t_{interval}	Suc.rate	Avg. Err.	δ
30	15	55	0.2	0.02	9	1	3.89322E-13	6.81674E-13
25	15	55	0.2	0.015	5	1	8.65132E-13	1.55604E-12
30	15	55	0.1	0.02	9	1	8.15837E-07	1.63167E-06
30	25	15	0.2	0.02	9	1	4.98201E-06	9.51869E-06
30	25	55	0.2	0.01	7	1	7.3299E-06	1.46598E-05
30	25	55	0.2	0.02	7	1	8.65236E-06	1.73047E-05
25	15	55	0.2	0.01	5	1	9.84034E-06	1.96807E-05
30	15	55	0.3	0.01	5	0.9	5.28488E-14	6.09323E-14
20	15	25	0.3	0.02	5	0.9	6.72776E-14	5.14056E-14
30	15	55	0.3	0.02	7	0.9	1.15345E-13	8.65234E-14
30	15	45	0.3	0.01	9	0.9	4.22961E-13	4.9642E-13
30	25	35	0.1	0.015	7	0.9	4.47037E-13	6.20104E-13
25	15	45	0.2	0.02	5	0.9	5.91155E-13	5.31764E-13
25	15	55	0.3	0.01	7	0.9	6.18099E-13	6.09032E-13
30	15	35	0.3	0.01	5	0.9	1.18239E-12	1.87658E-12
30	15	55	0.3	0.015	9	0.9	2.64346E-12	2.60625E-12
30	25	45	0.1	0.02	5	0.9	2.69456E-12	3.70756E-12
25	15	45	0.2	0.015	5	0.9	8.05097E-12	1.6029E-11
25	25	55	0.1	0.02	5	0.9	8.684E-12	1.26644E-11
25	15	25	0.2	0.01	7	0.9	3.10471E-09	6.20683E-09
25	15	35	0.3	0.015	9	0.9	9.53204E-09	1.90632E-08
25	15	55	0.1	0.02	7	0.9	1.96331E-08	3.92623E-08
30	15	15	0.2	0.01	7	0.9	3.16276E-08	5.47807E-08
25	15	35	0.3	0.015	7	0.9	3.27701E-08	6.55368E-08
25	15	25	0.1	0.015	7	0.9	4.42212E-08	8.84421E-08
25	15	45	0.2	0.015	9	0.9	9.39116E-08	1.87823E-07
20	15	15	0.1	0.02	5	0.9	1.01425E-07	2.02849E-07
25	25	55	0.2	0.01	5	0.9	8.87306E-07	1.14535E-06
30	55	55	0.1	0.015	9	0.9	1.28656E-06	2.22838E-06
30	25	25	0.2	0.02	7	0.9	1.36503E-06	2.3643E-06
30	35	35	0.1	0.01	5	0.9	1.48768E-06	2.57684E-06
25	15	25	0.3	0.01	9	0.9	1.79613E-06	3.59225E-06
25	15	55	0.3	0.015	5	0.9	3.49946E-06	6.99892E-06
25	15	15	0.1	0.01	9	0.9	3.56838E-06	7.13676E-06
25	25	25	0.1	0.02	7	0.9	3.73326E-06	3.80177E-06
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Table 6.2: Parameter tuning based on Branin and Shekel$_{4,7}$ functions

runs of five seconds. The success rate, average error and standard deviation are defined in the same way as it is for the parameter tuning in section 6.1. The test problems and optimal solutions given by appendix 9.2 are taken from the problems described by [14][2].

The Memetic Algorithm utilizes the Nelder–Mead simplex for local optimization at the end of each run. The results are compared to the Directed Tabu Search by [14] that also utilizes the Nelder–Mead simplex for local optimization. The success rate and average error over 100 runs is given for each of the five problems for both algorithms in table 6.3. Furthermore, the standard deviation is calculated for the Memetic Algorithm. The results cannot be compared directly since the running time of the DTS_{NMS} is not fixed[3]. Different stopping criterion related to the improvement of the algorithm are used instead. Thus, the DTS_{NMS} by [14] is allowed to use more computational time on the hard problems than the easier problems. However, the Memetic Algorithm can be considered converged after five seconds. In that way the results in table 6.3 can be seen as the *best* success rate and average error of the two metaheuristics. The success rates are seen to be very similar for the two algorithms for all functions except from the Easom function where the Memetic Algorithm has a success rate of 0.65 against 0.30 for the DTS_{NMS}. The average error is lower for the Memetic Algorithm than for the DTS_{NMS} in case of the $Shekel_{4,10}$ and the $Hartmann_{3,4}$ function. On the other hand the DTS_{NMS} performs better than the Memetic Algorithm in case of the $Shekel_{4,5}$, the $Hartmann_{6,4}$ and the Easom function. Comparing the mean success rate and mean average error over all the test problems the algorithms are seen to perform similar with a slight advantage to the Memetic Algorithm.

	MA_{NMS}			DTS_{NMS}	
Function	Suc.rate	Avg. Err.	δ	Suc.rate	Avg. Err.
$Shekel_{4,5}$	0.38	1.E-06	7.E-06	0.39	7.E-07
$Shekel_{4,10}$	0.23	4.E-08	2.E-07	0.22	1.E-05
$Hartman_{3,4}$	0.99	1.E-06	1.E-06	0.97	2.E-06
$Hartman_{6,4}$	0.64	3.E-06	1.E-05	0.68	2.E-06
Easom	0.65	7.E-06	2.E-05	0.30	5.E-09
	0.58	2.E-06	8.E-06	0.51	3.E-06

Table 6.3: Results from tests

In order to fulfill the condition of a successful trial in equation (6.1) on page 80, the objective value has to be close to the optimal value. In that way a

[2]See appendix 9.2 for a description of two inconsistencies

[3]Even if the computational time was fixed the results could not be directly compared if the tests were not performed on the same computer

successful run is very likely to be within the reach of the global optimum by means of local minimization. Thus, the success rate is considered the primary measure of performance in the tests since a low average error of the successful runs is a matter of letting the local minimization technique run for a longer time in order to further approach the optimal solution.

The results of the combined algorithm with a Simulated Annealing on top of the Memetic Algorithm are given by table 6.4. These results do not call for direct comparison with the results of the Memetic Algorithm itself. The success rate and the average error given by table 6.4 is based on five runs of the Memetic Algorithm. However, it should be noticed that it is the best five runs found by letting the Simulated Annealing search for the best parameters as described in section 5.2.3. Since the results can be seen as the best of a number of overall trials, it is not surprising that the success rates are higher than those of table 6.3. On the other hand the Simulated Annealing procedure is a substitution for the parameter tuning. It should be noticed that the results in table 6.4 do not rely on manual parameter tuning of the Memetic Algorithm prior to the tests.

	SA(MA_{NMS})	
Function	Suc.rate	Avg. Err.
$Shekel_{4,5}$	0.40	3.E-04
$Shekel_{4,10}$	0.80	7.E-05
$Hartman_{3,4}$	1.00	6.E-07
$Hartman_{6,4}$	1.00	4.E-11
Easom	1.00	0.E+00
	0.84	7.E-05

Table 6.4: Results from tests with SA on top of MA

The results obtained in this section should not be seen as an attempt to prove that the Memetic Algorithm with a divergent/convergent search strategy works better or worse than the Directed Tabu Search used for comparison. It is simply a matter of verifying that the algorithm works very well for non-linear continuous multi-modal functions. In section 6.3 an optimization problem within parameter calibration of a groundwater model is solved. The application to that particular real-life problem calls for time consuming runs of the algorithm because of the time consuming simulation model to predict the groundwater flow. Thus, it is preferable to evaluate the performance of the metaheuristic by mathematically defined functions first.

6.3 Groundwater Simulations

The case represented in this section is taken from groundwater modelling. The difference between the multi-modal continuous functions solved in section 6.2 and the multi-modal continuous real-life problem in this section is the evaluation process of each solution. Whereas each solution to e.g. the Branin function could be evaluated by a mathematically defined function implemented as part of the metaheuristic this is not the case here. The objective function is well defined as seen later in section 6.3.1.2. However, the objective function relies on the output from a simulation process depending on the trial solution. Thus, each evaluation cannot be described by a mathematically defined equation. It should rather be seen as a black box function since the details of the complicated simulation process in the external software does not need to be known. In other words, the relation between the solution input and the objective output is hidden by the simulation model.

6.3.1 Problem Overview

The case used in this thesis is based on a groundwater simulation model of the area (catchment) Hjordkær in Sønderjylland. The groundwater model is developed at *Niras Rådgivende ingeniører og planlæggere A/S* for their customer *Sønderjyllands Amt* and is available to the public. The model is developed for the Modflow-2000 simulation software using the GMS Modflow 5.1 interface for monitoring and manipulating the model.

The general aim of a groundwater model is to monitor groundwater resources as these can be difficult to observe in nature because *observation wells* have to be dug in order to measure the groundwater level, i.e. the *hydraulic head* given in meters. One of two complications by digging a well is the restrictions on where to set it up. The other problem is the influence caused by the well itself on the observations. In that sense the groundwater model substitutes manual observations that are difficult or even impossible to make.

The groundwater model used in this thesis simulates the hydraulic head (groundwater level) in twelve horizontal geological layers divided into a number of cells placed in a horizontal grid. Based on a simulation the different levels of hydraulic head can be monitored in each layer. An example of the hydraulic head levels in the Hjordkær catchment is given by the red curves in figure 6.1.

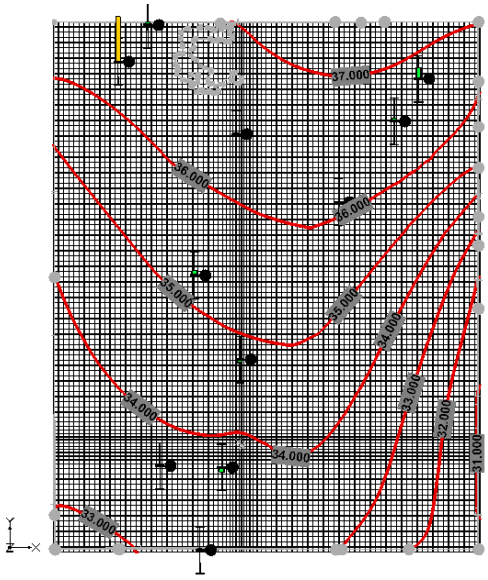


Figure 6.1: Groundwater monitoring provided by the model

6.3.1.1 Calibration Parameters

The groundwater simulation model is influenced by sixteen parameters in total. Specific for each of the twelve layers is a horizontal hydraulic conductivity K_h and a vertical hydraulic conductivity K_v, both given in m/s. In this model the horizontal hydraulic conductivity is specified by a parameter for each of the twelve layers. The vertical conductivity is linked through the *anisotropy* α which is a factor so that $\alpha = \frac{K_h}{K_v}$. The anisotropy is specified by two parameters in this model – one for layers containing sand α_{sand} and another for clay α_{clay}. Finally two parameters L_1 and L_2 called *lenses* are specified. The lenses describes the horizontal conductivity of a specific part of a layer. The calibration intervals for the sixteen parameters are given by table 6.5. The interval for the horizontal conductivity for the six layers containing clay differs from those containing sand. The intervals are based on those used for the manual calibration. However, the intervals for the automatic calibration are enlarged slightly to allow for a larger search space. It is seen that the lower limit for the horizontal conductivity of the layers containing sand actually equals the upper limit of the horizontal conductivity of clay. This, and the relatively large lower limit[4] (10^{-10}) of the horizontal conductivity of layers with clay is due to the

[4]When comparing to the values in the literature where the lower limit is likely to be 10^{-13}

fact that layers containing clay can be quite sandy in the area of Sønderjylland.

Parameter	Interval
$K_{1,\text{sand}} \dots K_{6,\text{sand}}$	$\in [10^{-6}; 10^{-3}]$
$K_{1,\text{clay}} \dots K_{6,\text{clay}}$	$\in [10^{-10}; 10^{-6}]$
α_{sand}	$\in [8; 10]$
α_{clay}	$\in [8; 10]$
L_1	$\in [10^{-6}; 10^{-3}]$
L_2	$\in [10^{-6}; 10^{-3}]$

Table 6.5: Intervals for calibration parameters

6.3.1.2 Objective Function

In order to calibrate the model a definition of how well the model performs has to be given. In this case an objective function in should measure the closeness of the simulated hydraulic head and the observed hydraulic head in a number of observation points. For this purpose data from eleven observation wells are accounted in the model. The placement of the wells are given by black dots in the example in figure 6.1, page 85. Two versions of the objective function are used. The simple version is an average of the absolute difference between the observed and simulated head given by equation (6.5) where the k an index describing the wells, x_k is the simulated hydraulic head at well k, and y_k is the observed hydraulic head at well k.

$$\text{Avg.Err.} = \frac{1}{N_{\text{obs}}} \sum_{k=1}^{N_{\text{obs}}} |x_k - y_k|, \quad N_{\text{obs}} = 11 \tag{6.5}$$

Another objective function that enforces greater punishment of relatively large gaps between simulated and observed values is the *root mean square error* (RMSE) which is used by [7]. However, where [7] describes the RMSE across different time steps it is only used across different observation wells here. The RMSE which is used as an objective function is given by equation (6.6).

$$\text{RMSE} = \sqrt{\frac{1}{N_{\text{obs}}} \sum_{k=1}^{N_{\text{obs}}} (x_k - y_k)^2}, \quad N_{\text{obs}} = 11 \tag{6.6}$$

6.3.1.3 Test Setup

For the mathematically defined functions each evaluation of a solution in the Memetic Algorithm only required a call of an internally defined objective function. The procedure is more complicated when an optimization of the parameters for the groundwater simulation is made. According to [13] the parameters are taken as input by Modflow-2000 through an LPF-package which reads an LPF-file as input. Furthermore, the observed and simulated hydraulic head in each observation well is written as output to a "_OS"-file by Modflow-2000 according to [15].

In order for the optimization to work the Memetic Algorithm needs to communicate with Modflow-2000 at each function evaluation. The procedure performed at each evaluation is illustrated by figure 6.2 where the steps given by the arrows in the clockwise loop are performed after each other. The sixteen parameter values are represented by an individual in the Memetic Algorithm. The parameters are written to the LPF-file by the Memetic Algorithm and the Modflow-2000 simulation software is started by the Memetic Algorithm. Modflow-2000 reads the parameters from the LPF-file. After simulation terminates the simulated and observed hydraulic head is written to the "_OS"-file which is taken as input by the evaluation function that calculates the objective value by a RMSE or an average error. The Memetic Algorithm waits for the Modflow-2000 simulation process to finish at each evaluation since the simulation can be seen as part of the evaluation process.

Each Modflow-2000 simulation is running the groundwater model which is given by several external files. The geological model setup defines amongst other things the number of layers, the thickness and whether it consists of sand or clay. Furthermore the Modflow-2000 software reads the observed hydraulic head values in each well from an external file as indicated by figure 6.2. The setup of the groundwater model used in this thesis is manipulated by the graphical GMS 5.1 Modflow interface. As described earlier the model is not developed by the author of this thesis and the geological setup is out of the scope of this thesis.

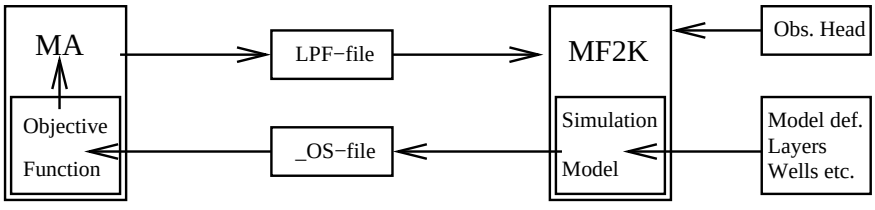


Figure 6.2: Test setup

6.3.2 Numerical Results and Interpretation

Each Modflow-2000 simulation is running for 40 to 150 seconds on the computer used for testing[5]. Due to the time consuming simulation process directly related to any evaluation of an individual in the Memetic Algorithm the maximum running time can obviously not be limited to five seconds. Instead the number of generations is limited to 60. To speed up the evolutionary process the population size is decreased. Thus, a μ value of nine is used whereas both λ_{mutate} and $\lambda_{\text{reproduce}}$ are set to seven. The maximum number of unsuccessful trials in the local search procedure is limited to 10. All other parameters are fixed to the values found by parameter tuning of the Branin and the $\text{Shekel}_{4,7}$ functions in section 6.1. The parameter values are given by equation (6.4), page 80.

Four tests are performed. Each test consists of one run of the Memetic Algorithm terminating after 60 generations and a local Nelder–Mead simplex optimization. The first two tests considers the data from ten observation wells. Since it was found that one observation well was not consistent with the remaining ten that observation well was discarded in the manual calibration. Thus, that specific well is also discarded for the objective function in two of the tests. The first objective function is the absolute average error between observed and simulated hydraulic head and the second objective is the RMSE of observed versus simulated hydraulic head. The third objective function considers the average error of all eleven wells whereas the fourth objective function is the RMSE over all wells. The objective values of the final solution in all four versions are compared to the objective value of the manually calibrated equivalent in table 6.6.

The first line of numbers in table 6.6 are based on the manually calibrated model. The average error and the RMSE over the ten observations and the eleven observations respectively are given by the values in bold. In the left side of the table the objective function is given for the automatic calibration.

The second line of numbers is based on optimization of the average error of the observation set including ten wells. In the second line of numbers both the average error and the RMSE is given based on the ten observation wells. In this case the average error is given in bold since it is used as the objective function. The RMSE is calculated based on the final solution. The bottom line indicates that the average error of 0.144 is an improvement of 3% compared to the manually calibrated model.

The third line of is based on the optimization of the RMSE of the observation set including ten wells. The value in bold is the objective value of the RMSE

[5]The processor used for testing the groundwater parameter estimation problem is an Intel Celeron M 1.5 GHz with 1024 MB RAM

and the calculated average error based on ten observations is given next to the RMSE. The RMSE of 0.182 is seen to be an improvement of 7% compared to the manually calibrated model. Furthermore, the calculated average error based on the solution obtained by using the RMSE as objective is given. The calculated average error turns out to be as good as when the average error itself was the objective.

The fourth line of numbers is based on an optimization where the average error based on eleven wells is the objective function. The objective value is given in bold and the calculated RMSE based on eleven observations is also given. This time the average error turns out to have increased by 5 % compared to the manually calibrated model.

In the fifth line the RMSE is used as objective function based on observations from eleven wells. The objective value of 0.595 given in bold is an improvement of 3% compared to that of the manually calibrated model. Furthermore, the calculated average error when using the RMSE as objective is seen to be below the average error when this was actually the objective. The fact that the RMSE is better when the average error is the objective can seem a bit strange. However, it should be noted that the optimization function which can be seen as a black box has multiple minima. By metaheuristic optimization it cannot be guaranteed which one of many good local minima is found. This depends on the search space and can even variate from one run to another.

The fact that using the RMSE as objective yields a lower average error than using the average error itself (and vice versa), might be explained by the different structures of the search space which could make one solution easier to find than another. On the other hand, the tests are based on only one run for each objective function. Thus, the properties of the two local minima could also be coincidental. The RMSE is seen to minimize the average error at least as much as the average error it self for both test sets. Thus, the results indicate that the RMSE provides a search space structure where the Memetic Algorithm finds low minima, more easily than for the average error objective.

The four solutions are given by the parameter values in table 6.7. From an optimization point of view the values might not be very interesting. However, a few conclusions can be drawn from the solutions. In the first sand layer and the first sand lens results given in **bold** show hydrological conductivities on the lower boundary of the solution space. This implies that the objective value might be lowered by enlarging the solution space for that layer. Hydrologically this could mean that the conductivity in the first layer is a bit lower than first anticipated. The second interesting observation is that all conductivities except one are above 10^{-8}, i.e. the lower part of the solution space of the conductivity of clay is not utilized. This can be interpreted as if the clay layers have a

	Method	Obj. function	10 Observations Avg. Err.	10 Observations RMSE	11 Observations Avg. Err.	11 Observations RMSE
1	Manual		**0.148**	**0.195**	**0.311**	**0.613**
2	Auto	$\text{Avg}_{10\text{ obs.}}$	**0.144**	0.209		
3	Auto	$\text{RMSE}_{10\text{ obs.}}$	0.144	**0.182**		
4	Auto	$\text{Avg}_{11\text{ obs.}}$			**0.325**	0.588
5	Auto	$\text{RMSE}_{11\text{ obs.}}$			0.315	**0.595**
		Improvement	**3%**	**7%**	**-5%**	**3%**

Table 6.6: Groundwater Modelling Results

large conductivity compared to the norm. However, this is not surprising when considering that real clay is relatively sparse in Sønderjylland.

Parameter/Objective	$\text{Avg}_{10\text{ obs.}}$	$\text{RMSE}_{10\text{ obs.}}$	$\text{Avg}_{11\text{ obs.}}$	$\text{RMSE}_{11\text{ obs.}}$
$K_{1,\text{sand}}$	**1.00E-6**	6.03E-4	1.90E-6	**1.00E-6**
$K_{2,\text{sand}}$	1.12E-4	4.88E-4	7.14E-4	6.61E-4
$K_{3,\text{sand}}$	6.65E-4	5.05E-4	3.73E-4	9.33E-4
$K_{4,\text{sand}}$	3.02E-4	3.09E-5	7.65E-5	5.81E-4
$K_{5,\text{sand}}$	2.13E-6	7.82E-5	5.30E-4	3.40E-5
$K_{6,\text{sand}}$	2.75E-5	1.65E-4	2.95E-4	3.31E-4
$K_{1,\text{clay}}$	4.47E-7	1.06E-7	3.77E-7	9.87E-7
$K_{2,\text{clay}}$	5.13E-8	7.93E-7	4.11E-7	2.49E-7
$K_{3,\text{clay}}$	5.36E-7	2.93E-7	6.40E-8	1.00E-10
$K_{4,\text{clay}}$	9.77E-7	9.34E-8	1.80E-8	2.41E-7
$K_{5,\text{clay}}$	7.79E-7	1.04E-7	4.52E-7	7.57E-7
$K_{6,\text{clay}}$	6.12E-7	1.33E-7	8.00E-7	3.69E-7
α_{sand}	9.03	8.34	8.30	9.20
α_{clay}	11.00	8.63	10.62	9.51
L_1	4.52E-5	2.14E-4	4.18E-4	**1.00E-6**
L_2	5.33E-5	1.23E-4	4.54E-5	2.18E-4

Table 6.7: Parameter values representing the four solutions

6.3.3 Concluding Remarks on Results

The improvements compared to the manually calibrated model shown by three out of four tests indicate that using the metaheuristic optimization is an advan-

tage in the area of groundwater modelling. The optimization procedure is in that way capable of replacing or assisting weeks of problematic manual calibration procedures. In case of the observation set based on ten wells the Memetic Algorithm even proved superior to the manual calibration.

Chapter 7

Discussion

The results based on optimization of the mathematically defined functions in section 6.2 can be seen as a verification that the algorithm yields competitive results. It is seen that the Memetic Algorithm by its divergent/convergent search strategy finds the optimal solution by at least as large a success rate and average error as the DTS_{NMS} by [14] which also utilizes a type of divergent/convergent search strategy.

The optimization of parameters for the groundwater problem shows that the Memetic Algorithm can be used for optimization of a black box problem. This kind of problem can be very complicated and can call for time consuming evaluations as it is seen in section 6.3. The results indicate that the automatic parameter optimization by the Memetic Algorithm can replace time consuming manual calibration. With one exception the solution found by the Memetic Algorithm is better than the solution found by manual calibration.

In case even better results are needed a two step approach could be taken. After one optimization run a number of parameters could be fixed. A second run would then be able to perform a more thorough metaheuristic search on the remaining parameters. For groundwater modellers, another approach could be to fix some of the parameters based on qualified assumptions. This would allow for a smaller search space which in the end makes the job of the metaheuristic easier. No matter what approach is used. The use of the metaheuristic optimization within

groundwater modelling will save time compared to the entirely manual approach.

Due to the time consuming simulation process a typical run of the algorithm takes three days when optimizing the groundwater simulation model. The key to shortening the running time is to allow for more groundwater simulations at the same time. Thus, parallelization would improve running time. Since it was possible to obtain the results without utilizing parallel programming no efforts has been made in this thesis to parallelize the algorithm. Further work could deal with this issue.

When considering that the groundwater simulation problem is a black box from the metaheuristics point of view, a number of other application areas appear. The black box problem could be anything in the range from econometric problems as presented by [17] to the adaptation of the *side winding locomotion* of a simulated *snake-like* robot presented by [23] or a third problem. Common to the problems that the Memetic Algorithm in this thesis directly apply to is the use of a continuous search space, i.e. the solution variables taken as input by the black box is defined in the continuous space. Furthermore, the output from the black box which is used to calculate the objective is also continuously defined. When the constraints on the solution variables are simply box constraints defining an upper and lower limit of the variables the algorithm is directly applicable. For more complicated constraints the metaheuristic implementation has to be adjusted.

The Memetic Algorithm is applied to a groundwater simulation model in 6.3. The results show that the metaheuristic is valuable for optimization of parameters in the groundwater simulation model. In three out of four cases the parameters found by the Memetic Algorithms outperforms those found by manual calibration. However, the real bonus is considered to be the amount of time saved for problematic manual calibration. A groundwater modeller can use the saved time for other tasks while the metaheuristic is determining the parameter values. As the Memetic Algorithm is basically capable of solving a black box problem it can also be applied to many other problem areas in the future.

Chapter 8

Conclusion

The problem solving process in this thesis utilizes both creative thinking and soft operations research methods. An algorithm is developed using creativity in the form of divergent and convergent thinking merged with metaheuristic optimization.

The purpose of this thesis is to design an algorithm capable of optimizing multimodal continuous functions. Even black box functions should be solved by the designed method. As seen from chapter 6 the results provide evidence that the final metaheuristic can both optimize mathematically defined multimodal functions and the parameter calibration of the groundwater simulation problem. The groundwater simulation model provide output that is optimized in a RMSE and average error objective function. The simulation itself is seen as a black box function. Time has only allowed for applying the metaheuristic method to the single area of groundwater modelling within what is considered black box functions. However, the metaheuristic method is not affected by the model behind the optimization problem. The only requirement for a simulation model within any application is the possibility of manipulating input parameters (e.g. by external files) from the metaheuristic implementation. Thus, several application areas should be possible. This part of the purpose is achieved.

Another aim of the thesis - stated in the introduction - is to merge the field of creative thinking and the use of soft operations research with metaheuristic

methods and mathematical optimization. By viewing the solution process it is seen that both rational thinking and creativity is used. The description of the metaheuristics is based on rational thinking whereas, the brainstorming on guiding tools for the search strategy introduces creativity. Furthermore, soft operations research methods are used in the form of a SWOT analysis for decision support. In this way creative thinking is used for the design process. Each part of the process can be seen as either a divergent or a convergent step as described by section 4.3. Thus, the aim of merging creative thinking and metaheuristic optimization has succeeded.

The last goal of utilizing creative thinking in the search strategy and implementation of the final algorithm has succeeded. The guiding tools affect the search strategy in a divergent and convergent way. The divergent steps are meant for escaping local minima whereas, the convergence allow for minima to be discovered. The switching between strategies is triggered partly by changes in the objective function, but also by the diversity of the population and an iteration counter. All together this make up a divergent/convergent strategy. However, other aspects of creative thinking could also be used in the search strategy for future setups.

Chapter 9

Evaluation

9.1 A Study of the Design Process

This section provides an insight in the creative thinking of the design process of this thesis. Parts of the process is viewed in retrospect on a higher level. The author of the thesis is viewed as a designer going through a process.

As described earlier the aim of this thesis has been to optimize multi-modal continuous functions amongst black box functions. Furthermore, the purpose has been to merge the area of creative thinking and soft operations research methodology with the mathematical optimization represented by the mathematical problems and the metaheuristic method. On this basis, at least two criteria for an acceptable outcome can be formulated for the project:

1. Design of an algorithm that is capable of optimizing continuous multi-modal functions even when the evaluation of the objective is considered a black box.
2. Utilization of creative thinking and soft operations research methods.

This formulation can be seen as a fixed criteria specified by an external *client* prior to the design process. With these criteria in mind, two important parts

of the design process is focused upon. In the following, the reasoning behind the design process is presented, followed by an interpretation of each of the two design strategies.

9.1.1 Choosing a Metaheuristic

A metaheuristic provides a way to search the solutions space. It requires the possibility to evaluate a solution by an objective function rather than the expression of the objective function itself. That metaheuristics work well for combinatorial problems does not necessarily mean that these are directly applied to continuous functions. Thus, a description of how to adapt various metaheuristics to continuous functions is given in this thesis. Besides finding out how to apply the metaheuristics, a detailed insight in the advantages of some metaheuristics over others is given. This is used to compare the metaheuristics.

In order to choose a metaheuristic further comparison is needed. A SWOT matrix is made for each metaheuristic. Apart from assisting in the design process the SWOT analysis also accommodates the criteria of using soft OR methods. In order to allow for creative divergent and convergent thinking in the search strategy of the algorithm the SWOT analysis is modified to focus on the opportunities in this area. Finally, the Memetic Algorithm is chosen based on its superior possibilities of a divergent/convergent search strategy.

9.1.1.1 Strategy

The strategy is described by applying a model presented by [5] to the design process. The process starts by the *highest goals* of the designer represented by the upper left corner of figure 9.1. In this case the use of a metaheuristic method is rather a high level goal of the designer than a direct demand given by the predefined criteria (purpose of the project). There is no evident conflict between the high level goal and one of the criteria. However, there is a potential conflict between the metaheuristic optimization method and the criteria of using or combining with soft OR methods. The problem is framed at the intermediate level by the designer. The algorithm has to work for continuous problems. Thus, all the metaheuristics are described for continuous functions. The framing is matched by the choice/design of one method capable of satisfying the predefined criteria. At the lower level a modified SWOT analysis is used as a *first principle* in order to bring the process from the problem frame in the left side to the solution concept in the right side of figure 9.1. Note, at the lower level there is no conflict between the SWOT analysis and the predefined criteria at the high

level.

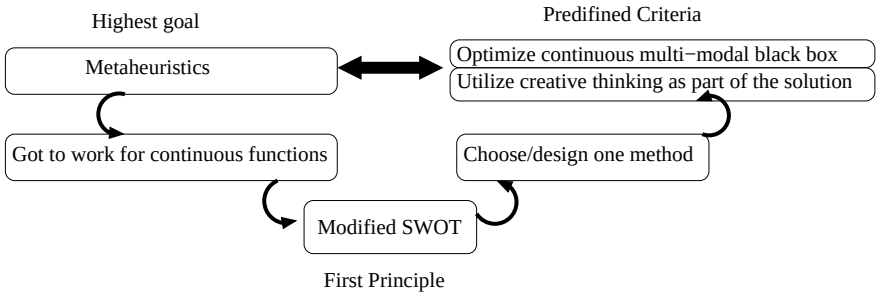


Figure 9.1: First part of the design process

9.1.2 Designing the Final Algorithm

The designing of the final algorithm is based on the Memetic Algorithm chosen earlier. The design should fulfill the predefined criteria of being able to optimize continuous multi-modal functions and utilize creative thinking. The main issue of the multi-modal functions is the possibility for the algorithm to get trapped in local minima. On the other hand, the search has to converge to a minimum by some strategy.

A creative process usually starts by divergent thinking followed by convergent thinking. In order to escape local minima creative thinking must be used in this form. The divergent steps make it possible to escape local minima whereas, the convergent steps approach minima. Thus, diversification and intensification is implemented in the final algorithm for guidance. The result not only meets the criteria of being able to optimize multi-modal functions. The implementation of guidance also accommodates the predefined criteria of merging the solution method with creative thinking.

9.1.2.1 Strategy

The designing of the final algorithm is initialized by the desire of the designer to use divergent/convergent thinking in the search strategy. This is represented by the upper left corner of figure 9.2. At first divergent/convergent thinking in the search strategy seems to be conflicting with the predefined criteria of designing

an algorithm for mathematical functions. However, in the intermediate level the problem is framed. The search has to be guided. At the lower level the first principle of guiding is used to escape local minima. The solution concept of implementing diversification and intensification is used to escape local minima and converge respectively as seen form the intermediate level of figure 9.2, right side.

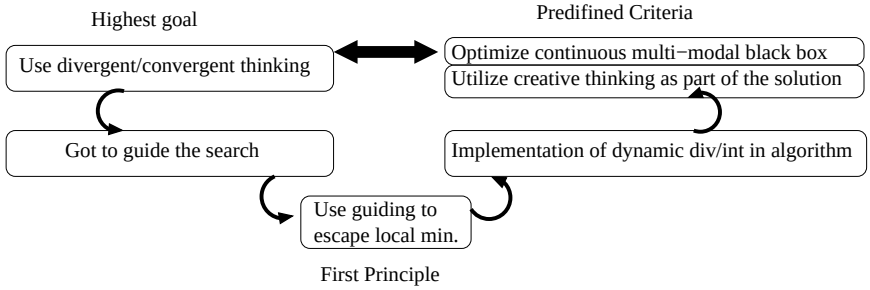


Figure 9.2: Second part of the design process

9.2 Evaluation of the Design Process

The design of the algorithm emerges from the potential conflict between the highest goal of the designer and the fundamental predefined criteria. Compared to the conflict presented in three case studies of professional designers by [5], the conflict in the design process of this project seems less evident. This can be explained partly by the following reasons:

1. The relatively long period of six months used for conducting the thesis. As a result the design process is broken down to more parts whereas, the two most important design strategies are represented in section 9.1. Each part of the design process emerges from the previous part. Thus, the highest goal of the designer at each stage of the design process develops. In that sense, the highest goal is influenced by what is possible and what is not in the previous part of the design process. The conflict between the highest goal and the predefined criteria is therefore less evident.

2. The design process of professional designers provide better understanding and insight in the process than the design process of a M.Sc. student. This is due to the cognitive strategy of trained designers might appear

more clear. The designers are professionals within their specific field of innovation whereas, a M.Sc. student can be considered an amateur in designing and in the application field.

3. Obvious similarities appear between the design processes described by [5] and the creative design in this thesis. However, a main difference is that the awareness of the process in this thesis is quite high whereas, professional designers focus mucm more on the product. The divergent and convergent steps are often caused by intuition and are not explicitly formulated. Furthermore, the purpose of the product is very specific in case of the three professionals. In the Master thesis the purpose is put together of more goals that are supposed to lead to an interesting project altogether.

The first point raises the question whether the creative design process is influenced negatively by having more time available. Think of another creative method, brainstorming is usually conducted over relatively few minutes in order to enhance spontaneous creativity. This is of course in contrast to rational thinking processes where more time usually contributes positively to the process and the outcome.

Bibliography

[1] George Bilchev and Ian C. Parmee. The Ant Colony Metaphor for Searching Continuous Design Spaces. In *Selected Papers from AISB Workshop on Evolutionary Computing*, pages 25–39, London, UK, 1995. Springer-Verlag. [20, 23]

[2] Pedro Castro Borges and René Victor Valqui Vidal. Fixed channel assignment in cellular communication systems considering the whole set of packed patterns. An investigation based on metaheuristics. *Control and Cybernetics*, 29, 2000. [6]

[3] Rachid Chelouah and Patrick Siarry. A hybrid method combining continuous tabu search and Nelder–Mead simplex algorithms for global optimization of multiminima functions. *European Journal of Operational Research*, 161, 2005. [12]

[4] Ling Chen, Jie Shen, Ling Qin, and Jin Fan. A Method for Solving Optimization Problem in Continuous Space Using Improved Ant Colony Algorithm. *Lecture Notes in Computer Science*, 3327, 2005. [23]

[5] Nigel Cross. Creative Thinking by Expert Designers. *The Journal of Design Research*, 4, 2004. [iv, vi, 98, 100, 101]

[6] Kathryn A. Dowsland. *Simulated Annealing*, pages 20–69. In [21], 1993. [7, 9]

[7] Jean-Philippe Drécourt and r. *Data asimilation in hydrological modelling*. PhD thesis, Technical University of Denmark, June 2004. [86]

[8] Emad Elbeltagi, Tarek Hegazy, and Donald Grierson. Comparison among five evolutionary-based optimization algorithms. *Advanced Engineering Informatics*, 19, 2005. [20, 35, 37, 39]

[9] Xiaoping Fan, Xiong Luo, Sheng Yi, Shengyue Yang, and Heng Zhang. Optimal Path Planning for Mobile Robots Based on Intensified Ant Colony Optimization Algorithm. pages 131–136. IEEE, 2003. [24]

[10] Yan Ge, Qing-Chun Meng, Chuan-Jun Yan, and Jing Xu. A Hybrid Ant Colony Algorithm for Global Optimization of Continuous Multi-Extreme Functions. pages 2427–2432. IEEE, 2004. [23]

[11] Fred Glover and Manuel Laguna. *Tabu Search*, pages 70–150. In [21], 1993. [19]

[12] David E. Goldberg. Genetic and Evolutionary Algorithms Come of Age. *Communications of the ACM*, 37, 1994. [25]

[13] Arlen W. Harbaugh, Edward R. Banta, Mary C. Hill, and Michael G. McDonald. Modflow-2000, The U.S. Geological Survey Modular Ground-water Model—User Guide to Modularization Concepts and the Ground-water Flow Process. Open-File Report 00-92, U.S. Geological Survey (USGS), 2000. [87]

[14] Abdel-Rahman Hedar and Masao Fukushima. Tabu Search directed by direct search methods for nonlinear optimization. *European Journal of Operational Research*, 170, 2006. [iii, v, 16, 17, 77, 80, 82, 93, I]

[15] Mary C. Hill, Edward R. Banta, Arlen W. Harbaugh, and Evan R. Anderman. Modflow-2000, The U.S. Geological Survey Modular Groundwater Model—User Guide to the Observation, Sensitivity, and Parameter-estimation Processes and Three Post-processing Programs. Open-File Report 00-184, U.S. Geological Survey (USGS), 2000. Prepared in cooperation with the U.S. Department of Energy. [87]

[16] Lester Ingber. Adaptive simulated annealing (ASA): lessons learned. *Control and Cybernetics*, 25, 1996. [9]

[17] Max E. Jerrell and Wendy A. Campione. Global Optimization of Econometric Functions. *Journal of Global Optimization*, 20, 2001. [2, 6, 9, 25, 33, 34, 94]

[18] Mingjun Ji and Huanwen Tang. Global optimizations and tabu search based on memory. *Applied Mathematics and Computation*, 159, 2004. [16, 18]

[19] Jeffrey C. Lagarias, James A. Reeds, Margaret H. Wright, and Paul E. Wright. Convergence Proporties of the Nelder–Mead Simplex Method in Low Dimensions. *SIAM Journal on Optimization*, 9, 1998. issue 1. [10, 14]

[20] Peter Merz. *Memetic Algorithms for Combinatorial Optimization Problems.* PhD thesis, Universität-Gesamthochschule Siegen, December 2000. [25, 29, 34, 35, 36, 40]

[21] Colin R. Reeves. *Modern Heuristics Techniques for Combinatorial Problems.* Blackwell Scientific Publications, 1993. [103, 104]

[22] Zhi-Guo Sun and Hong-Fei Teng. An Ant Colony Optimization based Layout Optimization Algorithm. pages 675–678. IEEE, 2002. [23]

[23] Ivan Tanev, Thomas Ray, and Andrzej Buller. Automated Evolutionary Design, Robustness, and Adaption of Sidewinding Locomotion of a Simulated Snake-Like Robot. *IEEE Transactions on Robotics*, 21, 2005. [94]

[24] Y. S. Teh and G. P. Rangaiah. Tabu search for global optimization of continuous functions with application to phase equilibrium calculations. *Computers and Chemical Engineering*, 27, 2003. [15, 16, 17, 19]

[25] Dušan Teodorović. Transport Modeling by Multi-agent Systems: A Swarm Intelligence Approach. *Transportation Planning and Technology*, 26, 2003. [19, 20]

[26] René Victor Valqui Vidal. Dealing with Problematic Situations. *Documentos de Trabajo en Análisis Económico, Economic Analysis Working Papers*, 4, 2005. [56]

[27] René Victor Valqui Vidal. The Future Workshop: Democratic problem solving. Informatics and Mathematical Modelling, Technical University of Denmark, 2005. [56]

[28] Bing Zhang, Dezhao Chen, and Weixiang Zhao. Iterative ant-colony algorithm and its application to dynamic optimization of chemical process. *Computer & Chemical Engineering*, 29, 2005. [23]

Index

Mathematically Defined Test Functions

Note that two inconsistencies appear between the functions used in this thesis and the functions given by [14]. The optimal solution and objective value of the $\text{Hartmann}_{6,4}$ function used in this thesis is set to the best solution found by 100 five-second-runs of the Memetic Algorithm, since the best value found is lower than the optimal value given by [14]. Furthermore, in order to get the right results the β values in any of the Shekel functions in this thesis are multiplied by $\frac{1}{4}$ compared to those of [14]. The objective value that is obtained by inserting the optimal solution given by [14] to the Shekel function given by [14] differs by almost a hundred percent from the objective value given by [14]. However, with the corrected β values the results are correct. This implies that it is a matter of a simple typing error.

.1 The Branin Function

Definition:

$$\text{Branin}(\mathbf{x}) = (x_2 - \frac{5}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6)^2 + 10(1 - \frac{1}{8\pi})\cos(x_1) + 10$$

Search space: $x_1 \in [-5; 10], x_2 \in [0; 15]$
Global minima: $\mathbf{x}^* = (-\pi, 12.275), (\pi, 2.275), (9.42478, 2.475)$
Objective value of global minima: $\text{Branin}(\mathbf{x}^*) = 0.397887$

.2 The Shekel$_{4,5}$ Function

Definition:

$$\text{Shekel}_{4,5}(\mathbf{x}) = -\sum_{j=1}^{5}\Big[\sum_{i=1}^{4}(x_i - C_{ij})^2 + \beta_j\Big]^{-1}, \beta = \frac{1}{40}[1,2,2,4,4]^T,$$

$$C = \begin{bmatrix} 4.0 & 1.0 & 8.0 & 6.0 & 3.0 \\ 4.0 & 1.0 & 8.0 & 6.0 & 7.0 \\ 4.0 & 1.0 & 8.0 & 6.0 & 3.0 \\ 4.0 & 1.0 & 8.0 & 6.0 & 7.0 \end{bmatrix}$$

Search space: $x_i \in [0; 10], i = 1 \ldots, 4$
Global Minimum: $\mathbf{x}^* = (4, 4, 4, 4)$
Objective value of global minimum: $\text{Shekel}_{4,5} = -10.1532$

.3 The Shekel$_{4,7}$ Function

Definition:

$$\text{Shekel}_{4,7}(\mathbf{x}) = -\sum_{j=1}^{7}\Big[\sum_{i=1}^{4}(x_i - C_{ij})^2 + \beta_j\Big]^{-1}, \beta = \frac{1}{40}[1,2,2,4,4,6,3]^T,$$

$$C = \begin{bmatrix} 4.0 & 1.0 & 8.0 & 6.0 & 3.0 & 2.0 & 5.0 \\ 4.0 & 1.0 & 8.0 & 6.0 & 7.0 & 9.0 & 5.0 \\ 4.0 & 1.0 & 8.0 & 6.0 & 3.0 & 2.0 & 3.0 \\ 4.0 & 1.0 & 8.0 & 6.0 & 7.0 & 9.0 & 3.0 \end{bmatrix}$$

Search space: $x_i \in [0; 10], i = 1 \ldots, 4$
Global Minimum: $\mathbf{x}^* = (4, 4, 4, 4)$
Objective value of global minimum: $\text{Shekel}_{4,7} = -10.4029$

.4 The Shekel$_{4,10}$ Function

Definition:

$$\text{Shekel}_{4,10}(\mathbf{x}) = -\sum_{j=1}^{10}\Big[\sum_{i=1}^{4}(x_i - C_{ij})^2 + \beta_j\Big]^{-1}, \beta = \frac{1}{40}[1,2,2,4,4,6,3,7,5,5]^T,$$

$$C = \begin{bmatrix} 4.0 & 1.0 & 8.0 & 6.0 & 3.0 & 2.0 & 5.0 & 8.0 & 6.0 & 7.0 \\ 4.0 & 1.0 & 8.0 & 6.0 & 7.0 & 9.0 & 5.0 & 1.0 & 2.0 & 3.6 \\ 4.0 & 1.0 & 8.0 & 6.0 & 3.0 & 2.0 & 3.0 & 8.0 & 6.0 & 7.0 \\ 4.0 & 1.0 & 8.0 & 6.0 & 7.0 & 9.0 & 3.0 & 1.0 & 2.0 & 3.6 \end{bmatrix}$$

Search space: $x_i \in [0; 10], i = 1 \ldots, 4$
Global Minimum: $\mathbf{x}^* = (4, 4, 4, 4)$
Objective value of global minimum: $\text{Shekel}_{4,10} = -10.5364$

.5 The Hartman$_{3,4}$ Function

Definition:

$$\text{Hartmann}_{3,4}(\mathbf{x}) = -\sum_{i=1}^{4}\alpha_i \exp\Big[-\sum_{j=1}^{3}A_{ij}(x_j - P_{ij})^2\Big], \alpha = [1, 1.2, 3, 3.2]^T,$$

$$A = \begin{bmatrix} 3.0 & 10 & 30 \\ 0.1 & 10 & 35 \\ 3.0 & 10 & 30 \\ 0.1 & 10 & 35 \end{bmatrix}, P = 10^{-4} \cdot \begin{bmatrix} 6890 & 1170 & 2673 \\ 4699 & 4387 & 7470 \\ 1091 & 8732 & 5547 \\ 381 & 5743 & 8828 \end{bmatrix}$$

Search space: $x_j \in [0; 1], j = 1, 2, 3$
Global minimum: $\mathbf{x}^* = (0.114614, 0.555649, 0.852547)$
Objective value of global minimum: $\text{Hartmann}_{3,4}(\mathbf{x}^*) = -3.86278$

.6 The Hartman$_{6,4}$ Function

Definition:

$$\text{Hartmann}_{6,4}(\mathbf{x}) = -\sum_{i=1}^{4}\alpha_i \exp\Big[-\sum_{j=1}^{6}B_{ij}(x_j - Q_{ij})^2\Big], \alpha = [1, 1.2, 3, 3.2]^T,$$

$$B = \begin{bmatrix} 10 & 3.0 & 17 & 3.05 & 1.78.0 & \\ 0.05 & 10 & 17 & 0.1 & 8.0 & 14 \\ 3.0 & 3.51.7 & 10 & 17 & 8.0 & \\ 17 & 8.0 & 0.05 & 10 & 0.1 & 14 \end{bmatrix}, Q = 10^{-4} \cdot \begin{bmatrix} 1312 & 1696 & 5569 & 124 & 8283 & 5886 \\ 2329 & 4135 & 8307 & 3736 & 1004 & 9991 \\ 2348 & 1451 & 3522 & 2883 & 3047 & 6650 \\ 4047 & 8828 & 8732 & 5743 & 1091 & 381 \end{bmatrix}$$

Search space: $x_j \in [0; 1], j = 1, \ldots, 6$
Global minimum: $\mathbf{x}^* = (0.200858, 0.150132, 0.478654, 0.276525, 0.311913, 0.65702)$
Objective value of global minimum: $\text{Hartmann}_{6,4}(\mathbf{x}^*) = -3.33539215295$

.7 The Easom Function

Definition:

$$\text{Easom}(\mathbf{x}) = -\cos(x_1)\cos(x_2)\exp(-(x_1 - \pi)^2 - (x_2 - \pi)^2) \qquad (1)$$

Search space: $x_i \in [-100; 100], i = 1, 2$
Global minimum: $\mathbf{x}^* = (\pi, \pi)$
Objective value of global minimum: $\text{Easom}(\mathbf{x}^*) = -1$

Parameter Tuning Results

.8 The Shekel$_{4,7}$ Function

μ	λ_{rep}	λ_{mut}	$\sigma_{\max}$	ϵ	t_{interval}	Suc.rate	Avg. Err.	δ
30	15	55	0.2	0.02	9	1	7.78644E-13	1.36E-12
30	15	15	0.2	0.01	7	1	1.50914E-12	2.97E-12
25	15	55	0.2	0.015	5	1	1.73026E-12	3.11E-12
30	15	55	0.1	0.02	9	1	1.63167E-06	3.26E-06
30	25	15	0.2	0.02	9	1	9.96402E-06	1.90E-05
30	25	55	0.2	0.01	7	1	1.46598E-05	2.93E-05
30	25	55	0.2	0.02	7	1	1.59565E-05	3.19E-05
25	15	55	0.2	0.01	5	1	1.96807E-05	3.94E-05
30	15	55	0.3	0.01	5	0.8	1.05698E-13	1.22E-13
25	15	45	0.2	0.015	5	0.8	1.24993E-13	1.03E-13
20	15	25	0.3	0.02	5	0.8	1.34555E-13	1.03E-13
30	15	55	0.3	0.02	7	0.8	2.3069E-13	1.73E-13
20	15	15	0.1	0.02	5	0.8	3.26826E-13	3.99E-13
25	15	35	0.3	0.015	7	0.8	4.03665E-13	5.23E-13
25	15	35	0.3	0.015	9	0.8	7.59349E-13	8.75E-13
30	15	45	0.3	0.01	9	0.8	8.45922E-13	9.93E-13
25	15	55	0.3	0.01	7	0.8	8.84342E-13	5.13E-13
30	25	35	0.1	0.015	7	0.8	8.94075E-13	1.24E-12
25	15	45	0.2	0.02	5	0.8	1.18231E-12	1.06E-12
25	15	15	0.3	0.015	5	0.8	1.19204E-12	1.23E-12
25	15	55	0.3	0.015	5	0.8	1.46115E-12	1.46E-12
30	15	15	0.1	0.015	9	0.8	1.615E-12	2.70E-12
25	15	45	0.2	0.015	9	0.8	1.64386E-12	2.05E-12
25	15	25	0.1	0.015	7	0.8	2.24936E-12	3.72E-12
30	15	35	0.3	0.01	5	0.8	2.36479E-12	3.75E-12
30	25	45	0.1	0.02	5	0.8	3.27799E-12	3.75E-12
25	15	15	0.1	0.01	9	0.8	4.05663E-12	5.25E-12
30	15	55	0.3	0.015	9	0.8	5.28692E-12	5.21E-12
25	15	25	0.3	0.01	9	0.8	1.01318E-11	1.21E-11
25	15	25	0.2	0.01	7	0.8	1.03432E-11	1.55E-11
25	25	55	0.1	0.02	5	0.8	1.71971E-11	2.50E-11
25	15	55	0.1	0.02	7	0.8	2.67425E-11	4.58E-11
25	25	55	0.2	0.01	5	0.8	1.75433E-06	2.25E-06
30	55	55	0.1	0.015	9	0.8	2.5731E-06	4.46E-06
30	25	25	0.2	0.02	7	0.8	2.73006E-06	4.73E-06
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Table 1: Parameter tuning based on the Shekel$_{4,7}$ function

.9 The Branin Function

μ	λ_{rep}	λ_{mut}	$\sigma_{\max}$	ϵ	t_{interval}	Suc.rate	Avg. Err.	δ
15	15	15	0.1	0.01	9	1	0	0.00E+00
15	15	15	0.3	0.015	5	1	0	0.00E+00
15	15	15	0.3	0.015	7	1	0	0.00E+00
15	15	25	0.1	0.01	5	1	0	0.00E+00
15	15	25	0.1	0.015	5	1	0	0.00E+00
15	15	25	0.1	0.015	9	1	0	0.00E+00
15	15	25	0.1	0.02	5	1	0	0.00E+00
15	15	25	0.1	0.02	7	1	0	0.00E+00
15	15	25	0.1	0.02	9	1	0	0.00E+00
15	15	25	0.2	0.015	7	1	0	0.00E+00
15	15	25	0.2	0.015	9	1	0	0.00E+00
15	15	25	0.3	0.015	5	1	0	0.00E+00
15	15	25	0.3	0.015	7	1	0	0.00E+00
15	15	25	0.3	0.02	5	1	0	0.00E+00
15	15	25	0.3	0.02	7	1	0	0.00E+00
15	15	35	0.1	0.01	9	1	0	0.00E+00
15	15	35	0.1	0.015	9	1	0	0.00E+00
15	15	35	0.1	0.02	7	1	0	0.00E+00
15	15	35	0.1	0.02	9	1	0	0.00E+00
15	15	35	0.2	0.01	5	1	0	0.00E+00
15	15	35	0.2	0.015	5	1	0	0.00E+00
15	15	35	0.3	0.015	9	1	0	0.00E+00
15	15	45	0.1	0.01	5	1	0	0.00E+00
15	15	45	0.1	0.01	7	1	0	0.00E+00
15	15	45	0.1	0.015	7	1	0	0.00E+00
15	15	45	0.2	0.02	5	1	0	0.00E+00
15	15	45	0.3	0.02	7	1	0	0.00E+00
15	15	45	0.3	0.02	9	1	0	0.00E+00
15	15	55	0.1	0.01	9	1	0	0.00E+00
15	15	55	0.1	0.015	7	1	0	0.00E+00
15	15	55	0.1	0.015	9	1	0	0.00E+00
15	15	55	0.2	0.01	5	1	0	0.00E+00
15	15	55	0.2	0.01	7	1	0	0.00E+00
15	15	55	0.2	0.015	9	1	0	0.00E+00
15	15	55	0.2	0.02	5	1	0	0.00E+00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Table 2: Parameter tuning based on the Branin function

Source Code of JAVA–Program

.10 The Main File (MainClass.java)

```
import java.util.Date;
import java.lang.*;

/*
 * The main class, execution starts here.
 */
class MainClass
{
    /*
     * main... ya know...                                          10
     */
    public static void main(String args[])
        {
            // Starting parameter tuning procedure
            // parameterTuning();
            //}

            // Starting simulated annealing
            //SA sim = new SA();
            //  sim.start(5000);                                    20
            //}


    System.out.println("Starting program");
    // optimal objective value for comparison
    double optVal= 0.397887; //-10.40294;
```

```
int runs = 1;
    double deviation;
    String testname;
    // logfilename                                                    30
    String logname = "Hartmann3_4";
    String [] probfiles = new String[10];
    double[] stat = new double[runs];
    double bestRun = 10;
    double avgSucces = 0;
    double runSucces = 0;
    double devSucces = 0;

    for (int y = 1; y<= 10; y++) {
        testname = logname+y+".txt";                                  40
        probfiles[y-1] = testname;
    }

    //      for(int i = 0; i < probfiles.length; i++){
    // Statistic filename
        testname = "statfile_hartmann" + ".log";
        MA meme = new MA(testname, "hartmann64.txt");
        meme.avgBest = 0;
        for(int r=0; r < runs; r++) {
            testname = probfiles[1] + "_progress_" + r + ".log";      50
            stat[r] = meme.start(new Date(), testname,1,5000,0.23,30,15,55,0.2,0.09,0.02,0.03,0.9,0.65, 0.1, 9);
            System.out.println(stat[r]);
        }
        deviation = 0;
        devSucces=0;
        avgSucces=0;
        runSucces=0;
        for(int r=0; r < runs; r++) {
            deviation += Math.pow(stat[r]-meme.avgBest/(double)runs,2);
            if (stat[r] < bestRun) {                                  60
                bestRun = stat[r];
            }
            //if(stat[r] < -3.3){
            if(Math.abs(optVal - stat[r]) < 0.0001*Math.abs(optVal) + 0.000001){
                avgSucces += stat[r];
                runSucces++;
            }
        }
        for(int r=0; r < runs; r++) {
            if(Math.abs(optVal - stat[r]) < 0.0001*Math.abs(optVal) + 0.000001){   70
                //if(stat[r] < -3.3){
                devSucces += Math.pow(stat[r]-avgSucces/(double)runSucces,2);
            }
        }
        devSucces = Math.sqrt(devSucces/runSucces);
        deviation = Math.sqrt(deviation/runs);
        LogFile statlog2 = new LogFile(logname+".log", true);
        statlog2.write(bestRun + " avgSuc= " + avgSucces/runSucces + " SucRate= "+ runSucces/runs
+ " devSuc= " + devSucces + " " + meme.avgBest/runs + " " + deviation + " " + meme.staIter/runs + "\n");
        statlog2.saveAndClose();                                      80
        // Futher statistical work is done in Excel
```

```
            meme.avgBest = 0;
            meme.staIter = 0;
            bestRun = 0;
        }

    // Parameter tuning procedure
    public static long parameterTuning(){
    System.out.println("Starting program");                                    90
    long runtime = 0; //parameterTuning();
    double optVal= 0.397887; //-10.40294;
        int runs = 5;
        double deviation;
        String testname;
        String logname = "Hartmann3_4";
        String [] probfiles = new String[10];
        double[] stat = new double[runs];
        double bestRun = 10;
        double avgSucces = 0;                                                  100
        double runSucces = 0;
        double devSucces = 0;
        for (int y = 1; y<= 10; y++) {
            testname = logname+y+".txt";
            probfiles[y-1] = testname;
        }
        testname = "statfile_hartmann" + ".log";
        MA meme = new MA(testname, "shekel47.txt");
        meme.avgBest = 0;
        int[] muArray = {15, 20, 25, 30};                                      110
        int[] lambdaRarray = {15, 25, 35, 45, 55};
        int[] lambdaMarray = {15, 25, 35, 45, 55};
        double[] sigmaMaxArray = { 0.1, 0.2, 0.3};
        double[] epsilonArray = {0.01, 0.015, 0.02};
        int[] iDivIntArray = {5,7,9};
        for(int f=0; f< muArray.length; f++){
            for(int g=0; g< lambdaRarray.length; g++){
                for(int h=0; h< lambdaMarray.length; h++){
                    for(int i=0; i< sigmaMaxArray.length; i++){
                        for(int j=0; j< epsilonArray.length; j++){             120
                            for(int k=0; k< iDivIntArray.length; k++){
                                // fixed double[] stepSize =
            for(int r=0; r < runs; r++) {
                testname = probfiles[1] + "_progress_" + r + ".log";
                stat[r] = meme.start(new Date(), testname,1,5000,0.23,muArray[f],lambdaRarray[g],
lambdaMarray[h],sigmaMaxArray[i],0.09,epsilonArray[j],0.03,0.9,0.65, 0.1, iDivIntArray[k]);
                System.out.println(stat[r]);
            }
            // reset statistics
            deviation = 0;                                                     130
            devSucces=0;
            avgSucces=0;
            runSucces=0;
            for(int r=0; r < runs; r++) {
                deviation += Math.pow(stat[r]-meme.avgBest/(double)runs,2);
                if (stat[r] < bestRun) {
```

```
                bestRun = stat[r];
            }
            //if(stat[r] < -3.3){
            if(Math.abs(optVal − stat[r]) < 0.0001*Math.abs(optVal) + 0.000001){
                avgSucces += stat[r];
                runSucces++;
            }
        }
        for(int r=0; r < runs; r++) {
            //if(stat[r] < -3.3){
            if(Math.abs(optVal − stat[r]) < 0.0001*Math.abs(optVal) + 0.000001){
                devSucces += Math.pow(stat[r]−avgSucces/(double)runSucces,2);
            }
        }
        devSucces = Math.sqrt(devSucces/runSucces);
        deviation = Math.sqrt(deviation/runs);
        LogFile statlog2 = new LogFile(logname+".log", true);
        statlog2.write(bestRun + " avgSuc= " + avgSucces/runSucces + " SucRate= "+ runSucces/runs
+ " devSuc= " + devSucces + " " + meme.avgBest/runs + " " + deviation + " " + meme.staIter/runs + " "
+ muArray[f]+ " " + lambdaRarray[g]+ " " +lambdaMarray[h]+ " " +sigmaMaxArray[i]+ " "
+epsilonArray[j]+ " " +iDivIntArray[k] + "\n");
        statlog2.saveAndClose();
        // Futher statistical work is done in Excel
        // Reset calculations
        meme.avgBest = 0;
        meme.staIter = 0;
        bestRun = 10;
                    }
                }
            }
        }
    }
    // Close all for loops
    return runtime;
  }
  //}

}
```

.11 The Memetic Algorithm (MA.java)

```
import java.util.*;
import java.lang.*;

public class MA
{
  //private LogFile progress;
  private DataObject data;
  private Random rand;
```

```
    private String statfilename;
                                                                                    10
    static public double avgBest = 0;
    static public int staIter = 0;

    /*
     * Create a new DataObject from the file named 'filename' and
     * initialize the pseudo random number generator
     */
    public MA(String statfile, String datafile)
    {
        data = new DataObject(datafile);                                            20
        rand = new Random();
        statfilename = statfile;
    }


    /*
     * Start the MA algorithm. Use 'runfilename' to log the
     * values of key variables in each iteration..
     */
    public double start(Date startTime, String runfilename, int iter, double maxRunTime, double alpha,  30
int my, int lambdaR, int lambdaM, double sigmaMax, double stepSize, double epsilon, double stepSizeInc,
double stepSizeDec, double epsilonDec, double minDiverse, int intervalDivInt)
    {

        // Read upper and lower bounds
        double bounds[][] = data.getBounds();
        // Adapt epsilon (minimum increment)
        epsilon = epsilon*(bounds[0][1]-bounds[0][0]);
        // define global step size for self adaptation of strategy parameter
        double tauM = alpha/(Math.sqrt(2*my));                                      40
        // define local step size for selfadaptation of strategy parameter
        double tau = alpha/(Math.sqrt(2*Math.sqrt(my)));
        double[][] bestSolution;
        int nIterations=0;
        //data.evaluations = 0;
        int noImp = 0;
        int stratPar= 8;
        //std::Vector<std::Vector<T> > myVector;
        double lambdaRbackup = lambdaR;
        double lambdaMbackup = lambdaM;                                             50
        Vector<double[][]> testPop = new Vector<double[][]>();
        Vector<double[][]> nextPop = new Vector<double[][]>();
        Vector<double[][]> offSpring = new Vector<double[][]>();
        Vector<double[][]> populationR = new Vector<double[][]>();
        Vector<double[][]> populationM = new Vector<double[][]>();
        Vector<double[][]> tempPopulation = new Vector<double[][]>();
        // Prepare the log file and initialize the population
        LogFile runlog = new LogFile(runfilename);
        LogFile poplog = new LogFile("popfile.m");
        int startPos=0;                                                             60
        double bestObj = 0;
        double[][] bestIndividual;
        bestIndividual = new double[3][data.getDimension()];
```

```
        // start the main MA loop
        Date now = new Date();
        // initial population
        double bestChange = 0;
        nextPop = initializeVector(my,sigmaMax,stepSize);
        bestObj = nextPop.firstElement()[2][0];
        bestSolution = nextPop.firstElement();                                    70
        while(now.getTime()−startTime.getTime() < maxRunTime) {
        //while(nIterations < 60){
            // Reproduction
            offSpring = reproduction(nextPop,lambdaR,my,stepSize);
            nextPop.addAll(offSpring);
            // Mutation
            //plotPop(poplog,nextPop);
            populationM = mutation(nextPop,lambdaM,stepSize,tauM,tau);
            nextPop.addAll(populationM);
            //plotPop(poplog,nextPop);                                            80
            // Selection
            nextPop = selection(nextPop,my,startPos);
            // Update LS step size
            stepSize = stepSize*stepSizeDec;
            // Update epsilon (least improvement)
            epsilon = epsilon*epsilonDec;

            // Calculate change in variables

            bestChange=0;                                                         90
            for(int i=0; i< data.getDimension(); i++){
                for(int k=1; k< 0.66*(my); k++){
                bestChange += Math.abs(bestSolution[0][i]−nextPop.elementAt(k)[0][i])/((bounds[i][1]−bounds[i][0]));
                }
                    }

            // Divergent / Convergent Properties (if new best is not very better than old best)
            //         old                    new
                    if((bestObj) <= (nextPop.firstElement()[2][0]) ){
                noImp=1;                                                          100
                stepSize = stepSize*(1.0+ stepSizeInc);
            }
            else{
                noImp=0;
            }

            // Update overall Best
            if((nextPop.firstElement()[2][0]) < bestObj){
                bestObj = (nextPop.firstElement()[2][0]);
                bestSolution = nextPop.firstElement();                            110
            }
            // (In Case no Real Improvement has occured the last x iterations)
            // Shift between diversification intensification
            //if(noImp == 1 || nIterations < 7){
            if(nIterations % (2*intervalDivInt) >= intervalDivInt || nIterations < 7){
                runlog.write("diversification " + nIterations++ + " " + bestObj + " "
+ (nextPop.firstElement()[2][0]) + " " + (nextPop.elementAt(my/2)[2][0]) + "\n");
                startPos = my;
```

```
            if(lambdaM < lambdaMbackup + 5){
                lambdaM++;
                lambdaR−−;
            }
        }
        else{
            runlog.write("intensification " + nIterations++ + " " + bestObj + " "
+ (nextPop.firstElement()[2][0]) + " " + (nextPop.elementAt(my/2)[2][0])+ "\n");
            startPos = 0;
            if(lambdaM > lambdaMbackup − 5){
                lambdaM−−;
                lambdaR++;
            }
        }
        // In Case population has converged
        if(bestChange < minDiverse || nextPop.firstElement()[2][0] >= nextPop.elementAt(my/2)[2][0] − epsilon){
            plotPop(poplog,nextPop);
            runlog.write("Meta-mutation \n");
            nextPop = metamutation(nextPop,stepSize,poplog);
            nextPop = selection(nextPop,my,0);
            plotPop(poplog,nextPop);
        }

        now = new Date();
    }
    // Create population of "dimension"+1 individuals for NM simplex
    Vector<double[][]> PopNM = new Vector<double[][]>();
    // Add best solution as vertex after being local searched ones
    double[][] firstIndividual = LocalSearch1(bestSolution,stepSize);
    PopNM.addElement(firstIndividual);
    // Add the next "dimension-2" solutions
    for(int l=1; l < data.getDimension(); l++){
        PopNM.addElement(nextPop.elementAt(l));
    }
    // Add last solution to PopNM
    int count=data.getDimension();
    while((nextPop.elementAt(count)[2][0]) <= (nextPop.firstElement()[2][0]) && count < my−1){
        count++;
    }
    //System.out.println(data.evaluations);
    PopNM.addElement(nextPop.elementAt(count));
    // Utilize Nelder-Mead
    bestIndividual = NM(PopNM,runlog);
    // Test the best point given in the report
    double[][] givenBest = new double[3][4];
    givenBest[0][0] = 4;
    givenBest[0][1] = 4;
    givenBest[0][2] = 4;
    givenBest[0][3] = 4;
    //givenBest[0][4] = 0.311652;
    //givenBest[0][5] = 0.657300;
    String txt = " ";
    for(int i = 0; i<data.getDimension(); i++){
        txt += " " + round(bestIndividual[0][i],6);
    }
```

```
    txt += "\n";
    for(int i = 0; i<data.getDimension(); i++){
        txt += " " + round(nextPop.firstElement()[0][i],6);
    }

    // Save and close the logfile
    poplog.saveAndClose();
    runlog.saveAndClose();

    bestObj = (bestIndividual[2][0]);
    // For parameter tuning and testing
    avgBest += bestObj;
    staIter += nIterations;

    // Append some stuff to the statistics file.
    LogFile statlog = new LogFile(statfilename, true);
    statlog.write(iter + " " + " " + " " + " " + (nIterations-1) + " " + bestObj + "\n");
    statlog.saveAndClose();

    return bestObj;
}

private void plotPop(LogFile poplog, Vector<double[][]> population){
    String varName = "st = [";
    for(int i = 0 ; i < data.getDimension(); i++){
        varName = "st" + varName;
        poplog.write(varName);
        for(int numero = 0; numero < population.size(); numero++){
            poplog.write(" " + population.elementAt(numero)[0][i]);
        }
        poplog.write("]; \n");
    }
    poplog.write("plot(stst,ststst,'o') \n \n");
}

private double[][] NM(Vector<double[][]> population, LogFile runlog){
    double ro = 1.0;
    double chi = 2.0;
    double gamma = 0.5;
    double sigma = 0.5;
    double epsilon = 0.0000001001;
    int counter = 0;
    int iDimension = data.getDimension();
    double[][] lastPoint;
    lastPoint = population.lastElement();
    double lastPointObj = (lastPoint)[2][0];
    double[][] nextLastPoint;
    nextLastPoint = population.firstElement();
    double[][] pointR;
    double[][] pointE;
    double[][] pointC;
    double[][] pointCC;
    double pointRobj = lastPointObj;
    double pointEobj = lastPointObj;
    double pointCobj = lastPointObj;
```

```
double pointCCobj = lastPointObj;
for(int q=0; q < population.size(); q++){                                    230
    runlog.write(" " + (population.elementAt(q)[2][0]) + " " + population.size() + " " + iDimension + "\n");
}
while(nextLastPoint[2][0] < ((lastPoint[2][0]) − epsilon )){
runlog.write("inde her ! \n");
lastPoint = population.firstElement();
for(int h=0;h<30;h++){
    for(int q=0; q < population.size(); q++){
        runlog.write(" " + (population.elementAt(q)[2][0]) + " " + population.size() + " \n");
    }
    runlog.write("Counter = "+ counter + "\n");                              240
    pointR = reflect(population,ro);
    pointRobj = pointR[2][0];
    if(pointRobj < (population.elementAt(iDimension−1)[2][0])){
        if(pointRobj >= (population.firstElement()[2][0])){
            population = accept(population,pointR);
            runlog.write("accept reflected point \n");
        }
        else{
            pointE = expand(population,pointR,chi);
            pointEobj = pointE[2][0];                                        250
            if(pointEobj < pointRobj){
                population = accept(population,pointE);
                runlog.write("accept expanded point"+ "\n");
            }
            else{
                population = accept(population,pointR);
                runlog.write("accepted reflected point R less than E"+ "\n");
            }
        }
    }                                                                        260
    else{
        if(pointRobj < (population.elementAt(iDimension)[2][0])){
            pointC = contractOut(population,pointR,gamma);
            pointCobj = pointC[2][0];
            if(pointCobj <= pointRobj){
                population = accept(population,pointC);
                runlog.write("Accept contracted point on the outside"+ "\n");
            }
            else{
                population = shrink(population,sigma);                       270
                runlog.write("Shrink POP = no accepted points (out)"+ "\n");
            }
        }
        else{
            pointCC = contractIn(population,gamma);
            pointCCobj = (pointCC[2][0]);
            if(pointCCobj < (population.elementAt(iDimension)[2][0])){
                population = accept(population,pointCC);
                runlog.write("Accept contracted point on the inside"+ "\n");
            }                                                                280
            else{
                population = shrink(population,sigma);
                runlog.write("Shrink POP = no accepted points (in)"+ "\n");
```

```
                    }
                }
            }
            counter++;
        }
        nextLastPoint = population.firstElement();
        }
        runlog.write("R" + (population.firstElement()[2][0]) + "\n");
        return population.firstElement();
    }

    private Vector<double[][]> accept(Vector<double[][]> population, double[][] newVertex){
        int indexMax=population.size()-2;
    population.removeElementAt(indexMax+1);
    double newVertexObj = (newVertex[2][0]);
    for(int i=indexMax; i >= 0; i--){
        if(newVertexObj >= (population.elementAt(i)[2][0])){
            population.insertElementAt(newVertex,i+1);
            return population;
        }
        else if(i==0){
            population.insertElementAt(newVertex,0);
            return population;
        }
    }
    //population.insertElementAt(newVertex,0);
    return population;
}

private Vector<double[][]> shrinkAccept(Vector<double[][]> population, double[][] newVertex){
    int indexMax=population.size();
    double newVertexObj = (newVertex[2][0]);
    for(int i=0; i < indexMax; i++){
        if(newVertexObj < (population.elementAt(i)[2][0])){
            population.insertElementAt(newVertex,i);
            return population;
        }
        else if(i==indexMax-1){
            population.addElement(newVertex);
            return population;
        }
    }
    //population.addElement(newVertex);
    return population;
}

private double[][] reflect(Vector<double[][]> population, double ro){
    int indexMax=population.size()-1;
    int iDimension = data.getDimension();
    double[][] centroid = new double[3][iDimension];
    double[][] pointR = new double[3][iDimension];
    double[][] lastVertex = population.lastElement();
    for(int j=0; j < indexMax; j++){
        for(int i=0; i < iDimension; i++){
            centroid[0][i] += (population.elementAt(j)[0][i])/((double)indexMax);
```

```
            }
    }
    for(int i=0; i < iDimension; i++){
        pointR[0][i] = centroid[0][i] + ro*(centroid[0][i]−lastVertex[0][i]);
    }
    pointR[2][0] = data.evalSolution(pointR);
    return pointR;
}

private double[][] expand(Vector<double[][]> population, double[][] pointR, double chi){
     int indexMax=population.size()−1;
     int iDimension = data.getDimension();
     double[][] centroid = new double[3][iDimension];
     double[][] pointE = new double[3][iDimension];
     for(int j=0; j < indexMax; j++){
        for(int i=0; i < iDimension; i++){
            centroid[0][i] += (population.elementAt(j)[0][i])/((double)indexMax);
            }
    }
    for(int i=0; i < iDimension; i++){
        pointE[0][i] = centroid[0][i] + chi*(pointR[0][i]−centroid[0][i]);
    }
    pointE[2][0] = data.evalSolution(pointE);
    return pointE;
}

private double[][] contractOut(Vector<double[][]> population, double[][] pointR, double gamma){
     int indexMax=population.size()−1;
     int iDimension = data.getDimension();
     double[][] centroid = new double[3][iDimension];
     double[][] pointC = new double[3][iDimension];
     for(int j=0; j < indexMax; j++){
        for(int i=0; i < iDimension; i++){
            centroid[0][i] += (population.elementAt(j)[0][i])/((double)indexMax);
            }
    }
    for(int i=0; i < iDimension; i++){
        pointC[0][i] = centroid[0][i] + gamma*(pointR[0][i]−centroid[0][i]);
    }
    pointC[2][0] = data.evalSolution(pointC);
    return pointC;
}

private double[][] contractIn(Vector<double[][]> population, double gamma){
     int indexMax=population.size()−1;
     int iDimension = data.getDimension();
     double[][] lastVertex = population.lastElement();
     double[][] centroid = new double[3][iDimension];
     double[][] pointCC = new double[3][iDimension];
     for(int j=0; j < indexMax; j++){
        for(int i=0; i < iDimension; i++){
            centroid[0][i] += (population.elementAt(j)[0][i])/((double)indexMax);
            }
    }
    for(int i=0; i < iDimension; i++){
```

```
        pointCC[0][i] = centroid[0][i] − gamma*(centroid[0][i]−lastVertex[0][i]);
    }
    pointCC[2][0] = data.evalSolution(pointCC);
    return pointCC;
}

private Vector<double[][]> shrink(Vector<double[][]> population, double sigma){
    int indexMax=population.size();
    int iDimension = data.getDimension();
    Vector<double[][]> nextPop = new Vector<double[][]>();
    double[][] firstVertex = population.firstElement();
    nextPop.addElement(firstVertex);
    double[][] newVertex = new double[3][iDimension];
    for(int j=1; j < indexMax; j++){
        for(int i=0; i<iDimension ; i++){
            newVertex[0][i] = firstVertex[0][i] + sigma*(population.elementAt(j)[0][i]−firstVertex[0][i]);
        }
        newVertex[2][0] = data.evalSolution(newVertex);
        nextPop=shrinkAccept(nextPop,newVertex);
    }
    return nextPop;
}

    public static double round(double val, int places) {
        long factor = (long)Math.pow(10,places);

        // Shift the decimal the correct number of places
        // to the right.
        val = val * factor;

        // Round to the nearest integer.
        long tmp = Math.round(val);

        // Shift the decimal the correct number of places
        // back to the left.
        return (double)tmp / factor;
    }

    private Vector<double[][]> selection(Vector<double[][]> population, int mu,int startPos){
        Vector<double[][]> SelPop = new Vector<double[][]>();
        int iDimension = data.getDimension();
        //Create mutated population of
        double[][] candidate;
        int canIndex = 0;
        candidate = new double[3][iDimension];
        for(int k=0; k<mu; k++){
            candidate = population.elementAt(startPos);
            canIndex=startPos;
            for(int l=startPos+1; l<population.size(); l++){
                if((population.elementAt(l)[2][0]) < (candidate[2][0])){
                    candidate = population.elementAt(l);
                    canIndex = l;
                }
            }
            SelPop.addElement(candidate);
```

```
            population.removeElementAt(canIndex);
        }
        return SelPop;
    }

    private Vector<double[][]> metamutation(Vector<double[][]> population, double stepSize, LogFile poplog){
        Vector<double[][]> populationM = new Vector<double[][]>();
        int iDimension = data.getDimension();
        //Create mutated population of
        populationM.addElement(population.firstElement());
        double[][] individual;
        individual = new double[3][iDimension];
        for(int l=1; l<population.size(); l++){
                individual = mutate2(population.elementAt(l));
            for(int h=0; h<2; h++){
                individual = LocalSearch1(individual,stepSize);
            }
            populationM.addElement(individual);
        }
        plotPop(poplog,populationM);
        return populationM;
    }

    private Vector<double[][]> mutation(Vector<double[][]> population, int lambdaM,
double stepSize, double tauM, double tau){
        Vector<double[][]> populationM = new Vector<double[][]>();
        int iDimension = data.getDimension();
        //Create mutated population of
        double[][] individual;
        individual = new double[3][iDimension];
        for(int l=0; l<lambdaM; l++){
            individual = population.elementAt(rand.nextInt(population.size()));
            individual = mutate(individual, tauM, tau);
            for(int h=0; h<1; h++){
                individual = LocalSearch1(individual,stepSize);
            }
            populationM.addElement(individual);
        }
        return populationM;
    }

    private double[][] mutate(double[][] individual, double tauLocal, double tauGlobal){
        int iDimension = data.getDimension();
        double[][] mutatedIndividual = new double[3][iDimension];
        double randVar= rand.nextGaussian();
        double randVar2 = rand.nextGaussian();
        for(int i=0; i<iDimension; i++){
            mutatedIndividual[1][i]= individual[1][i];
            // Mutation of Gene
            mutatedIndividual[0][i]= individual[0][i] + (mutatedIndividual[1][i])*randVar;
        }
        mutatedIndividual[2][0] = data.evalSolution(mutatedIndividual);
        return mutatedIndividual;
    }
```

```
private double[][] mutate2(double[][] individual){
    int iDimension = data.getDimension();
    for(int i=0; i<iDimension; i++){
        individual[0][i]= individual[0][i] + 6*(individual[1][i])*(Math.random()−0.5);
    }
    individual[2][0] = data.evalSolution(individual);
    return individual;                                                          510
}

private Vector<double[][]> reproduction(Vector<double[][]> population, int lambdaR, int mu, double stepSize){
    Vector<double[][]> populationR = new Vector<double[][]>();
    int iDimension = data.getDimension();
    //Create population of offspring
    double[][] individual;
    individual = new double[3][iDimension];
    for(int l=0; l<lambdaR; l++){
        int parent1 = rand.nextInt(mu);                                         520
        int parent2 = rand.nextInt(mu);
        individual = crossOver1(population.elementAt(parent1),population.elementAt(parent2));
        for(int h=0; h<1; h++){
            individual = LocalSearch1(individual,stepSize);
        }
        populationR.addElement(individual);
    }
    return populationR;
}
                                                                                530
private double[][] crossOver1(double[][] parent1, double[][] parent2){
    int iDimension = data.getDimension();
    double[][] offspring = new double[3][iDimension];
    //offspring = parent1;
    for(int i=0; i<iDimension; i++){
        if(Math.random() < 0.5){
            offspring[0][i] = parent2[0][i];
            offspring[1][i] = parent2[1][i];
        }
        else{                                                                   540
            offspring[0][i] = parent1[0][i];
            offspring[1][i] = parent1[1][i];
        }
    }
    offspring[2][0] = data.evalSolution(offspring);
    return offspring;
        //return parent1;
}

private Vector<double[][]> initializeVector(int popSize, double sigmaMax, double stepSize){  550
    Vector<double[][]> population = new Vector<double[][]>();
    int iDimension = data.getDimension();
    double bounds[][] = data.getBounds();
    double[][] individual;
    individual = new double[3][iDimension];
    for(int k=0; k<popSize; k++){
        for(int i=0; i<iDimension; i++){
            individual[0][i] = bounds[i][0] + Math.random()*(bounds[i][1]−bounds[i][0]);
```

```
            individual[1][i] = (bounds[i][1]−bounds[i][0])*sigmaMax*Math.random();
        }
        individual[2][0] = data.evalSolution(individual);
        for(int h=0; h<1; h++){
        individual = LocalSearch1(individual,stepSize);
        }
        population.addElement(individual);
    }
        return population;
}

private double[][] LocalSearch1(double[][] individual, double stepSize){
    double stepSizeBackup = stepSize;
    double bounds[][] = data.getBounds();
    double[][] candidate;
    int iDimension = data.getDimension();
    candidate = new double[3][iDimension];
    // First LS step
    for(int i=0; i<iDimension; i++){
        // Assure Individual is not out of bounds
        if(individual[0][i] > bounds[i][1]){
            individual[0][i] = bounds[i][1];
        }
        if(individual[0][i] < bounds[i][0]){
            individual[0][i] = bounds[i][0];
        }
        // Make a candidate
        candidate[0][i] = individual[0][i] + (bounds[i][1]−bounds[i][0])*stepSize*(Math.random()−0.5);
        // Assure candidate is within search space
        if(candidate[0][i] > bounds[i][1]){
            candidate[0][i] = bounds[i][1];
        }
        if(candidate[0][i] < bounds[i][0]){
            candidate[0][i] = bounds[i][0];
        }

        candidate[1][i] = individual[1][i];
    }
    candidate[2][0] = data.evalSolution(candidate);
    int counter = 0;
    // Following LS steps
    while((candidate[2][0]) >= (individual[2][0]) && counter < 50){
     for(int i=0; i<iDimension; i++){
         // make candidate
        candidate[0][i] = individual[0][i] + (bounds[i][1]−bounds[i][0])*stepSize*(Math.random()−0.5);
        // make sure candidate is in search space
        if(candidate[0][i] > bounds[i][1]){
            candidate[0][i] = bounds[i][1];
        }
        if(candidate[0][i] < bounds[i][0]){
            candidate[0][i] = bounds[i][0];
        }
     }
     candidate[2][0] = data.evalSolution(candidate);
     counter++;
```

```
        stepSize = stepSize*0.6;
       }
       stepSize=stepSizeBackup;
       if(counter >= 50){
           return individual;
       }
       return candidate;
   }

}
```

.12 The Data Input and Evaluation Object (DataObject.java)

```
import java.io.*;

/*
 * The data object contains the problem data to be worked on.
 */
class DataObject
{
    private int nItems, capacity;
    private int[] FixedWeightArray;
    private double[][] ProfitMatrix;
    //private int[][] ProfitMatrix;
    private StreamTokenizer st;
    private boolean endOfLine = false;

    //    public int iterator = 0;
    private double[][] MatrixA;
    private double[] VectorAlpha;
    private double[][] MatrixP;
    private int iDimension;
    private int jDimension;
    private double lBound;
    private double uBound;

    /*
     * Init data object based on the data in 'filename'
     */

    public DataObject(String filename)
    {
        //readAndParseFile2(filename);
        // This is only for Branin specifically
        jDimension = 16;
    }

    /*     Bounds for Branin function
```

```
public double[][] getBounds()
{
    double temp[][];
    temp = new double[jDimension][2];
    temp[0][0] = -5.0;
    temp[0][1] = 10.0;
    temp[1][0] = 0.0;
    temp[1][1] = 15.0;
    return temp;
}
*/
// Bounds for hydraulic conductivity for ground water simulation
public double[][] getBounds(){
    double temp[][];
    temp = new double[jDimension][2];
    temp[0][0] = 0.000001; // HK_101 Sand
    temp[0][1] = 0.001;
    temp[1][0] = 8.0; // VANI_201
    temp[1][1] = 12.0;
    temp[2][0] = 8.0; // VANI_202
    temp[2][1] = 12.0;
    temp[3][0] = 0.000001; // HK_103 Sand
    temp[3][1] = 0.001;
    temp[4][0] = 0.000001; // HK_303
    temp[4][1] = 0.001;
    temp[5][0] = 0.000001; // HK_105 Sand
    temp[5][1] = 0.001;
    temp[6][0] = 0.0000000001; // HK_104 Ler
    temp[6][1] = 0.000001;
    temp[7][0] = 0.000001; // HK_305
    temp[7][1] = 0.001;
    temp[8][0] = 0.000001; // HK_107 Sand
    temp[8][1] = 0.001;
    temp[9][0] = 0.0000000001; // HK_106 Ler
    temp[9][1] = 0.000001;
    temp[10][0] = 0.0000000001; // HK_111 Ler
    temp[10][1] = 0.000001;
    temp[11][0] = 0.000001; // HK_112 Sand
    temp[11][1] = 0.001;
    temp[12][0] = 0.0000000001; // HK_110 Ler
    temp[12][1] = 0.000001;
    temp[13][0] = 0.000001; // HK_109 Sand
    temp[13][1] = 0.001;
    temp[14][0] = 0.0000000001; // HK_108 Ler
    temp[14][1] = 0.000001;
    temp[15][0] = 0.0000000001; // HK_113 Ler
    temp[15][1] = 0.000001;
    return temp;
}

public int getDimension()
{
    jDimension = 16;
    return jDimension;
}
```

```
    /* Bounds for Hartmann and Shekel functions
    public double[][] getBounds()
    {
        double temp[][];
        temp = new double[jDimension][2];
        for(int i=0; i < jDimension; i++){
        temp[i][0] = lBound;
        temp[i][1] = uBound;
        }                                                              100
        return temp;
    }

    public int getDimension()
    {
    return jDimension;
    }
    */
      /*
       * evaluates the solution and returns the corresponding profit.   110
       * An infeasible solution is indicated by a return value of
       *  "infinity".
       */


    // Easom
    public double evalSolution2(double[][] solution){
        double objective =0;
        objective = −(Math.cos(solution[0][0]))*Math.cos(solution[0][1])
*Math.exp(−Math.pow((solution[0][0]−Math.PI),2)−Math.pow((solution[0][1]−Math.PI),2));  120
        return objective;
    }


    // eval som for Branin functions

    public double evalSolution3(double[][] solution)
    {
        double objective = 0;
        objective = Math.pow(solution[0][1]−(5/(4*Math.pow(Math.PI,2)))*Math.pow(solution[0][0],2)  130
+(5/Math.PI)*solution[0][0] − 6,2) + 10*(1−1/(8*Math.PI))*Math.cos(solution[0][0]) + 10;
        return objective;
    }

    // Evaluate Shekel functions
    public double evalSolution4(double[][] solution)
    {
        double objective = 0;
        for(int i=0; i<iDimension; i++)
            {                                                          140
                double evalSum = 0;
                for(int j=0; j<jDimension; j++)
                    {
                        evalSum += (Math.pow((solution[0][j]−MatrixA[i][j]),2) + VectorAlpha[i]) ;
                    }
```

```
            objective += Math.pow(evalSum,−1);
        }
    return −objective;
}
                                                                                    150
  // eval sol for Hartmann functions
public double evalSolution5(double[][] solution)
{
    double objective = 0;
    for(int i=0; i<iDimension; i++)
        {
            double evalSum = 0;
            for(int j=0; j<jDimension; j++)
                {
                    evalSum = evalSum + MatrixA[i][j]*Math.pow((solution[0][j]−MatrixP[i][j]),2) ; 160
                }
            objective = objective − VectorAlpha[i]*Math.exp(−evalSum);
        }
    return objective;
}

// evaluates solution by RMSE or average gap between obs and simulated values
    public double evalSolution(double[][] solution){
        double RMSE = 0.0;
        double[][] observations;                                                    170
        // Delete old backup
        String s =null;
        File f1 = new File("lpf_backup.txt");
        File f2 = new File("hjordkaer_20lag_5.lpf");
        boolean success = f2.renameTo(f1);
        if(!success){
            System.out.println("can't rename");
        }
        // (Copy lpf file to lpf_backup)
        // Write the parameters to the parameter file                               180
        writeParameterFile("hjordkaer_20lag_5.lpf", solution);
        // Delete lpf_backup
        File f3 = new File("lpf_backup.txt");
        f3.delete();
        // Run the MODFLOW Simulation
        System.out.println("Starting a simulation");
        //data.toPrint(solution);
        try{
            Process p = Runtime.getRuntime().exec("mf2k hjordkaer_20lag_5.mfn"); //Step 1 and 2
            BufferedReader stdInput = new BufferedReader(new InputStreamReader(p.getInputStream())); 190
            BufferedReader stdError = new BufferedReader(new InputStreamReader(p.getErrorStream()));
            System.out.println("here:\n");
            while((s = stdInput.readLine()) != null){
                System.out.println(s);
            }
            System.out.println("errors:\n");
            while((s = stdError.readLine()) != null){
                System.out.println(s);
            }
            p.waitFor(); // wait for external program to close                      200
```

```
            stdInput.close();
            stdError.close();
            p.destroy();
            System.out.println("Simulation OK");
        }catch (IOException e) {
            System.out.println("Error - Cannot simulate " + e);
        }
        catch(InterruptedException e){
        System.out.println("Cannot wait " + e);
        }                                                                    210
        // Read the output i.e. the simulated head and the observed head
        observations = readObsFile("hjordkaer_20lag_5._os");
        // Calculate the RMSE based on the two head observations
        int nrOfObs = observations[0].length;
        int dim = observations.length;
        for(int nr=0; nr < nrOfObs; nr++){
            if(observations[1][nr] == 1.0){
                observations[1][nr] = 10.0;
            }
            //if(nr != 1){                                                   220
            RMSE += Math.pow((observations[0][nr]−observations[1][nr]),2);
            //RMSE += Math.abs(observations[0][nr]-observations[1][nr]);
            //}
        }
        RMSE = Math.sqrt(RMSE/(nrOfObs));
        //RMSE = RMSE/(nrOfObs);
        System.out.println("Obj = " + RMSE);
        return RMSE;
    }
                                                                             230

public void writeParameterFile(String filename, double[][] solution)
 {
    LogFile fileCopy = new LogFile(filename);
    double temp;
    String fileReader = "o";
    String line;
    String lineStart;
    String lineEnd;
    int counter = 0;                                                         240
    DataInputStream dis = null;
    /* Manually calibrated solution
    double[][] solution = new double[3][jDimension];
    solution[0][0] = 0.0008;
    solution[0][1] = 10.0;
    solution[0][2] = 10.0;
    solution[0][3] = 0.0008;
    solution[0][4] = 0.0002;
    solution[0][5] = 0.0008;
    solution[0][6] = 0.0000005;                                              250
    solution[0][7] = 0.00005;
    solution[0][8] = 0.00001;
    solution[0][9] = 0.00000001;
    solution[0][10] =0.00000001;
    solution[0][11] =0.00001;
```

```
solution[0][12] =0.00000001;
solution[0][13] =0.00001;
solution[0][14] =0.00000001;
solution[0][15] =0.00000001;
*/
//      solution[16] =1;
//solution[17] =1;

// Open the file
//openFile2("lpf_backup.txt");
try{
    File f = new File("lpf_backup.txt");
    FileInputStream fis = new FileInputStream(f);
    BufferedInputStream bis = new BufferedInputStream(fis);
    dis = new DataInputStream(bis);
} catch(IOException e){
    System.err.println("Cannot read file");
}

fileReader = nextLine(dis);
while(fileReader != null){
    if(fileReader.startsWith("HK_")){
        lineStart = fileReader.substring(0,10);
        lineEnd = fileReader.substring(fileReader.length()-2,fileReader.length());
        if(counter < jDimension){
            line = String.valueOf(solution[0][counter]);
            }
        else{
            line = (fileReader.substring(10,fileReader.length()-2)).trim();
        }
        if(!lineEnd.startsWith(" ")){
            line += " ";
        }
        fileCopy.write(lineStart + line + lineEnd + "\n");
        counter++;
    }
    else{
        if(fileReader.startsWith("VANI_")){
            lineStart = fileReader.substring(0,14);
            lineEnd = fileReader.substring(fileReader.length()-2,fileReader.length());
            if(counter < jDimension){
                line = String.valueOf(solution[0][counter]);
            }
            else{
                line = (fileReader.substring(14,fileReader.length()-2)).trim();
            }
            if(!lineEnd.startsWith(" ")){
                line += " ";
            }
            //System.out.println(lineStart + line + lineEnd);
            fileCopy.write(lineStart + line + lineEnd + "\n");
            counter++;
        }
        else{
            fileCopy.write(fileReader + "\n");
```

```
            }
        }
        fileReader = nextLine(dis);
    }
    //Close the updated file
    fileCopy.saveAndClose();
    //Close the original file
        try{
            dis.close();
        } catch(IOException ioe){
        }
}

public double[][] readObsFile(String filename){
    String fileReader = "o";
    double temp;
    int nrOfObs = 11;
    DataInputStream dis = null;
    // Open the file
    try{
        File f = new File(filename);
        FileInputStream fis = new FileInputStream(f);
        BufferedInputStream bis = new BufferedInputStream(fis);
        dis = new DataInputStream(bis);
    } catch(IOException e){
        System.err.println("Cannot read file");
    }

    // allocate observation matrix
    double[][] headSimObs = new double[2][nrOfObs];

    for(int nr=0; nr < nrOfObs ; nr++){
        // Read next line
        fileReader = nextLine(dis);
        try{
        headSimObs[0][nr] = Double.valueOf((fileReader.substring(1,18)).trim()).doubleValue();
        headSimObs[1][nr] = Double.valueOf((fileReader.substring(18,34)).trim()).doubleValue();}
        catch(IndexOutOfBoundsException e){
            System.out.println("string not long enough");
        }
    }
    try{
        dis.close();
    } catch(IOException ioe){
    }
    toPrint(headSimObs);
    return headSimObs;
}

private void toPrint(double[][] headSimObs){
    int nrOfObs = headSimObs[0].length;
    int dim = headSimObs.length;
    for(int nr=0; nr < nrOfObs ; nr++){
        for(int i=0; i < dim; i++){
            System.out.print(headSimObs[i][nr] + " ");
```

```
        }
        System.out.print("\n");
    }
}

//* This function is able to read data for Hartmann functions
 //
    private void readAndParseFile2(String filename)
{
    double temp;

    // Open the file
    openFile(filename);

    // Set end-of-line as tokens
    st.eolIsSignificant(true);

    while (endOfLine != true) {
        nextNumber();
    }

    // Unset end-of-line as tokens
    st.eolIsSignificant(false);

    iDimension = (int)nextNumber();

    jDimension = (int)nextNumber();

    lBound = (double)nextNumber();

    uBound = (double)nextNumber();

    // Allocate memory.
    MatrixA = new double[iDimension][jDimension];
    MatrixP = new double[iDimension][jDimension];
    VectorAlpha= new double[iDimension];

    for(int i=0; i<iDimension; i++) {
        for(int j=0; j<jDimension; j++) {
        MatrixA[i][j] = (double)nextNumber();
        }
    }

    for(int i=0; i<iDimension; i++) {
        for(int j=0; j<jDimension; j++) {
        MatrixP[i][j] = (double)((double)nextNumber()/10000);
        }
    }
    for(int i=0; i<iDimension; i++) {
        VectorAlpha[i] = (double)nextNumber();

    }
    //System.out.println(toString());
}
```

```
    /*
     * Get the next number from the open file
     */

    private String nextLine(DataInputStream dis){
        String line = "EOF";
        try{line=dis.readLine();}
        catch(IOException e)
            {                                                                    430
                System.err.println("Failed to get next line");
            }
        return line;
    }

    private double nextNumber()
    {
        try{ st.nextToken(); }
        catch(IOException e)
            {                                                                    440
                System.err.println("Error: Failed to get next number. Is the file open?");
            }
        if(st.ttype == StreamTokenizer.TT_EOF)
            {
                System.err.println("Error: End of file reached");
            }
        if(st.ttype == StreamTokenizer.TT_EOL)
            {
                endOfLine = true;
            }                                                                    450
        return st.nval;
    }

  private String nextWord()
   {
        try{ st.nextToken(); }
        catch(IOException e)
            {
                System.err.println("Error: Failed to get next number. Is the file open?");
            }                                                                    460
        if(st.ttype == StreamTokenizer.TT_EOF)
            {
                System.err.println("Error: End of file reached");
                return "9.9999";
            }
        if(st.ttype == StreamTokenizer.TT_EOL)
            {
                endOfLine = true;
            }
        if(st.ttype == StreamTokenizer.TT_NUMBER){                               470
            return String.valueOf(st.nval);
        }
        if(st.ttype == StreamTokenizer.TT_WORD){
            return st.sval;
        }
```

```
        return "\n";
    }
    /*
     * Open the file and create a StreamTokenizer so we can get tokens from the file.
     */
    private void openFile(String filename)
    {
        try{ st = new StreamTokenizer(new FileReader(filename)); }
        catch(FileNotFoundException e)
            {
                System.err.println("File \"" + filename + "\" not found");
                System.exit(0);
            }
    }


    /*
     * Convert the data object to a string. Used as a debug function to help
     * check and visualize the matrix. Quite slow when displaying lots of data.
     */
    public String toString()
    {
        String output = iDimension + " x " + jDimension + "\n";
        output += "The search space is bounded by the following box constraints: \n";
        output += "Lower Bound=" + lBound + " Upper Bound=" + uBound + "\n";
        output += "\n A= \n";

        // matrix
        for(int i=0; i<iDimension; i++)
            {
                for(int j=0; j<jDimension; j++)
                    output += " " + MatrixA[i][j];
                output += "\n";
            }
        output += "\n P= \n";
        for(int i=0; i<iDimension; i++)
            {
                for(int j=0; j<jDimension; j++)
                    output += " " + MatrixP[i][j];
                output += "\n";
            }
        output += "\n alpha= ";
        for(int i=0; i<iDimension; i++)
            {
                output += " " + VectorAlpha[i];
            }
        return output;
    }
}
```

.13 The Documentation File (LogFile.java)

```
import java.io.*;

/*
 * This is a logfile class
 */
class LogFile
{
        private BufferedWriter file;
        private String filename;

        /*
         * Open a file called 'filename', ready for writing.
         */
        public LogFile(String filename)
        {
            openFile(filename, false);
        }

        /*
         * Open a file called 'filename', ready for writing/appending.
         */
        public LogFile(String filename, boolean append)
        {
            openFile(filename, append);
        }

        /*
         * Open a file called 'filname', ready for writing/appending.
         * If the append flag is false the file is truncated when opened,
         * if it is true, writing will start at the end of the file.
         */
        private void openFile(String filename, boolean append)
        {
                try
                {
                        this.filename = filename;
                        file = new BufferedWriter(new FileWriter(filename, append));
                }
                catch(IOException ioe)
                {
                        System.err.println("Error: Unable to open file \"" + filename + "\"");
                }
        }

        /*
         * Flush the write buffer and close the file.
         */
```

```
        public void saveAndClose()
        {
                try
                {
                        file.close();
                }
                catch(IOException ioe)
                {
                        System.err.println("Error: Unable to close file \"" + filename + "\"");
                }
        }


        /*
         * Write 'str' to the open file.
         */
        public void write(String str)
        {
                try
                {
                        file.write(str);
                }
                catch(IOException ioe)
                {
                        System.err.println("Error: Unable to write to file \"" + filename + "\"");
                }
        }
}
```

.14 The Simulated Annealing (SA.java)

```
import java.util.*;
import java.lang.*;

public class SA
{
    //private LogFile progress;
    private DataObject data;
    private Random rand;
    private String statfilename;

    static public double avgBest = 0;
    static public int staIter = 0;

    /*
     */
    public SA()
    {
        rand = new Random();
    }
```

```
                                                                                    20

   public void start(double maxRT){
      System.out.println("started");
       LogFile SAlog = new LogFile("sa_progress.log");
       System.out.println("Starting program" + (int)(2.6));
       //long runtime = parameterTuning();
       double[][] ulLimits= {
           { 0.15, 0.25},      // 1 alpha
           { 15.0, 30.0},      // 2 mu
           { 15.0, 55.0},     // 3 lambda R                                        30
           { 15.0, 55.0},     // 4 lambda M
           { 0.1, 0.3},        // 5 sigmaMax
           { 0.09, 0.09},       // 6 stepSize
           { 0.01, 0.02},     // 7 epsilon
           { 0.03, 0.09},      // 8 Step size increase in percent
           { 0.8, 0.9},       // 9 Step size decrease in percent
           { 0.4, 0.7},        // 10 epsilon decrease in percent
           { 0.1, 0.1},       // 11 Minimum relative difference of a converged population
           { 5.0, 9.0}};         // 12 Interval between intensify/diversify
                                                                                    40
       // Temperature
       double temp = 100.0;
       // Step size d
       double d = 0.1;
       // Treshold
       double tresHold = 0.5;
       // percentage update of temperature
       double alpha=0.9;
       // percentage update of stepsize
       double beta=0.99;                                                            50
       // Number of allowed iteration without improvement
       int nR = 10;
       // initial solution
       double[] solution = new double[ulLimits.length];
       for(int i=0; i < ulLimits.length; i++){
           solution[i] = ulLimits[i][0] + Math.random()*(ulLimits[i][1]−ulLimits[i][0]);
       }
       double[] currentObj = getObj(maxRT, solution[0], (int)solution[1], (int)solution[2], (int)solution[3],
solution[4], solution[5], solution[6], solution[7], solution[8], solution[9], solution[10], (int)solution[11]);
       // Candidate solution                                                        60
       double[] candidate = new double[ulLimits.length];
       // candidate objective
       double[] candObj = currentObj;
       // Best solution
       double[] bestSol = solution;
       // Vector of last nR best objectives
       Vector<double[]> bestSolObj = new Vector<double[]>();
       bestSolObj.addElement((currentObj));
       while(temp > tresHold){
           System.out.println("her " + bestSolObj.size());                          70
           while((double)bestSolObj.firstElement()[0] > (double)bestSolObj.lastElement()[0] || bestSolObj.size() < 10){
               System.out.println("ups");
               SAlog.write((double)bestSolObj.lastElement()[0] + " " + (double)bestSolObj.lastElement()[1]
+ " " + currentObj[0] + " " + bestSol[0] + " " + (int)bestSol[1] + " " + (int)bestSol[2] + " "
```

```
+ (int)bestSol[3] + " " + bestSol[4] + " " + bestSol[5] + " " + bestSol[6] + " " + bestSol[7] + " "
+ bestSol[8] + " " + bestSol[9] + " " + bestSol[10] + " " + (int)bestSol[11] + "\n" );
            // Make new candidate
            for(int i=0; i<ulLimits.length;i++){
                candidate[i] = solution[i] + d*(Math.random()−0.5)*(ulLimits[i][1]−ulLimits[i][0]);
                // make sure candidate is within allowed range                                    80
                if(candidate[i] > ulLimits[i][1]){
                    candidate[i] = ulLimits[i][1];
                }
                if(candidate[i] < ulLimits[i][0]){
                    candidate[i] = ulLimits[i][0];
                }

            }
            candObj = getObj(maxRT, candidate[0], (int)candidate[1], (int)candidate[2], (int)candidate[3],
candidate[4], candidate[5], candidate[6], candidate[7], candidate[8], candidate[9], candidate[10], (int)candidate[11]); 90
            // Check if candidate is better than current BEST
            if(candObj[0] < (Double)bestSolObj.lastElement()[0]){
                bestSolObj.addElement((candObj));
                bestSol=candidate;
            }
            else{
                // Otherwise best sol (and) obj remains the same
                bestSolObj.addElement(bestSolObj.lastElement());
            }
            if(candObj[0] < currentObj[0]){                                                     100
                currentObj = candObj;
                solution = candidate;
            }
            else{
                if(Math.random() < 0.1*Math.exp((currentObj[0]−candObj[0])/temp)){
                    currentObj = candObj;
                    solution = candidate;
                }
            }
            while(bestSolObj.size() > 10){                                                      110
                bestSolObj.removeElementAt(0);
            }
        }
        currentObj = bestSolObj.lastElement();
        solution = bestSol;
        temp = temp*alpha;
        d = d*beta;
        bestSolObj = new Vector<double[]>();
        bestSolObj.addElement((currentObj));
    }                                                                                           120

    SAlog.saveAndClose();
    System.out.println(currentObj);
}

  private double[] getObj(double maxRunTime, double alpha, int my, int lambdaR, int lambdaM,
double sigmaMax, double stepSize, double epsilon, double stepSizeInc, double stepSizeDec,
double epsilonDec, double minDiverse, int intervalDivInt){
    double optVal = −3.335352;
```

```
        int runs = 5;
        double deviation;
        String testname;
        String logname = "Hartmann3_4";
        String [] probfiles = new String[10];
        double[] stat = new double[runs];
        double bestRun = 100;
        double runSucces = 0;

        for (int y = 1; y<= 10; y++) {
            testname = logname+y+".txt";
            probfiles[y-1] = testname;
        }

        //      for(int i = 0; i < probfiles.length; i++){
            testname = "statfile_hartmann" + ".log";
            MA meme = new MA(testname, "hartmann64.txt");
            meme.avgBest = 0;
            for(int r=0; r < runs; r++) {
                testname = probfiles[1] + "_progress_" + r + ".log";
                //stat[r] = meme.start(new Date(), testname,1,0.18,1500,30,25,25,0.2,0.09,0.01,0.03,0.985,0.5, 0.05, 8);
                stat[r] = meme.start(new Date(), testname, 1, maxRunTime, alpha, my, lambdaR,
lambdaM, sigmaMax, stepSize, epsilon, stepSizeInc, stepSizeDec, epsilonDec, minDiverse, intervalDivInt);
            }
            runSucces=0;
            deviation = 0;
            for(int r=0; r < runs; r++) {
                deviation += Math.pow(stat[r]-meme.avgBest/(double)runs,2);
                if (stat[r] < bestRun) {
                    bestRun = stat[r];
                }
                if(Math.abs(optVal - stat[r]) < 0.0001*Math.abs(optVal) + 0.000001){
                    runSucces++;
                }
            }
            deviation = Math.sqrt(deviation/runs);
            LogFile statlog2 = new LogFile(logname+".log", true);
            statlog2.write(bestRun + " " + meme.avgBest/runs + " " +
                           deviation + " " + meme.staIter/runs + "\n");
            statlog2.saveAndClose();
            // Futher statistical work is done in Excel

            // meme.avgBest = 0;
            meme.staIter = 0;
            //bestRun = 0;
            //return bestRun
            double[] retVal = new double[2];
            retVal[0] = meme.avgBest/runs;
            retVal[1] = runSucces/runs;
            return retVal;
    }
   }
```

Example of Progress Logfile

```
diversification 0 -3.681262028487361 -3.681262028487361 -1.7693714445311552
diversification 1 -3.681262028487361 -3.638973210671149 -2.5147728504834497
diversification 2 -3.796259787322114 -3.796259787322114 -2.8838142332329126
diversification 3 -3.796259787322114 -3.7902787722085796 -3.4135538747983865
diversification 4 -3.8040448460099507 -3.8040448460099507 -3.5395915118995354
diversification 5 -3.832261160787402 -3.832261160787402 -3.7141782990881627
diversification 6 -3.8328090841937184 -3.8328090841937184 -3.8071914541760634
intensification 7 -3.8443415261397744 -3.8443415261397744 -3.814033961087362
intensification 8 -3.8443415261397744 -3.8443415261397744 -3.8345540873888613
diversification 9 -3.8471125006811606 -3.8471125006811606 -3.838976171364568
diversification 10 -3.8472130431959046 -3.8472130431959046 -3.832363544040683
diversification 11 -3.847215678227104 -3.847215678227104 -3.8333466715092763
diversification 12 -3.847215678227104 -3.8466727333308896 -3.829188554083683
diversification 13 -3.8477911709298596 -3.8477911709298596 -3.8339432527064328
diversification 14 -3.8482309415219462 -3.8482309415219462 -3.823665263519541
diversification 15 -3.853259195954801 -3.853259195954801 -3.823647442097851
diversification 16 -3.853259195954801 -3.8481795062222566 -3.82137686530419
diversification 17 -3.853259195954801 -3.8495740556911495 -3.8009416931129554
intensification 18 -3.853259195954801 -3.8494593539261466 -3.8102567660558497
intensification 19 -3.853259195954801 -3.8494655213033795 -3.831739690394253
intensification 20 -3.853259195954801 -3.8494655213033795 -3.8373803794131005
intensification 21 -3.853259195954801 -3.8494655213033795 -3.8395206649444957
intensification 22 -3.853259195954801 -3.8494655213033795 -3.8410926302269117
intensification 23 -3.853259195954801 -3.8494655213033795 -3.843922672841333
intensification 24 -3.853259195954801 -3.8508404439727433 -3.8457725530789464
intensification 25 -3.853259195954801 -3.8508404439727433 -3.8479499635418857
intensification 26 -3.853259195954801 -3.851159715414616 -3.849260963117441
diversification 27 -3.853259195954801 -3.851683082342535 -3.8496937505522277
diversification 28 -3.853259195954801 -3.850934761761458 -3.84931352100595
diversification 29 -3.853259195954801 -3.8528694019354695 -3.8436506815214315
diversification 30 -3.853259195954801 -3.8529479185629714 -3.8431700433671763
diversification 31 -3.853259195954801 -3.852078706672963 -3.838374064205776
diversification 32 -3.853259195954801 -3.8524239197036434 -3.8375940222816913
diversification 33 -3.853259195954801 -3.8516091032842543 -3.838854819678817
diversification 34 -3.853259195954801 -3.851995751907993 -3.8298101071552666
diversification 35 -3.853259195954801 -3.8523856752612455 -3.8301439932876686
intensification 36 -3.8549799554971873 -3.8549799554971873 -3.8084354198808024
intensification 37 -3.8564962598147585 -3.8564962598147585 -3.847692624745898
intensification 38 -3.8564962598147585 -3.8564962598147585 -3.8499566272388566
intensification 39 -3.8564962598147585 -3.8564962598147585 -3.8510780035169994
intensification 40 -3.8564962598147585 -3.8564962598147585 -3.85199953942076
intensification 41 -3.8577250539214543 -3.8577250539214543 -3.8528313521811084
intensification 42 -3.8577253090299477 -3.8577253090299477 -3.8550243220770226
intensification 43 -3.859200986352953 -3.859200986352953 -3.856477512648243
intensification 44 -3.859200986352953 -3.859200986352953 -3.8571739659829656
diversification 45 -3.859200986352953 -3.859200986352953 -3.857725176259774
diversification 46 -3.859200986352953 -3.8589896775317474 -3.8574194540755005
diversification 47 -3.859200986352953 -3.859133882563256 -3.8524925712428155
diversification 48 -3.8594702310955804 -3.8594702310955804 -3.853071165178392
diversification 49 -3.8594702310955804 -3.858268359647633 -3.8524874112859875
diversification 50 -3.8605068953775885 -3.8605068953775885 -3.853037085720738
diversification 51 -3.8605068953775885 -3.8588491984567543 -3.844197641071765
diversification 52 -3.8605068953775885 -3.857759576399526 -3.842075210087038
```

```
diversification 53 -3.861322498542325 -3.861322498542325 -3.8499594896623113
intensification 54 -3.861322498542325 -3.861187559638028 -3.8474875819415453
intensification 55 -3.861322498542325 -3.861187559638028 -3.856405820962126
intensification 56 -3.86186604086796 -3.86186604086796 -3.857660841981647
intensification 57 -3.8620552560388886 -3.8620552560388886 -3.8597359106375584
intensification 58 -3.8620552560388886 -3.8620552560388886 -3.860471965143944
intensification 59 -3.862530353041458 -3.862530353041458 -3.8610058366776965
intensification 60 -3.862530353041458 -3.862530353041458 -3.861601477595949
intensification 61 -3.862674469567937 -3.862674469567937 -3.8620130175660714
intensification 62 -3.862695339148372 -3.862695339148372 -3.862308714474091
Meta-mutation
diversification 63 -3.862695339148372 -3.862695339148372 -3.8156873673764458
diversification 64 -3.862699505950367 -3.862699505950367 -3.8346789555072087
diversification 65 -3.862699505950367 -3.853564476528409 -3.822162631378787
diversification 66 -3.862699505950367 -3.850850909878009 -3.8242222305782003
diversification 67 -3.862699505950367 -3.858748997594951 -3.83254333395067
diversification 68 -3.862699505950367 -3.856465195804008 -3.8385976331151204
diversification 69 -3.862699505950367 -3.8562771819912807 -3.834010050935867
diversification 70 -3.862699505950367 -3.856476488855182 -3.83269090787001
diversification 71 -3.862699505950367 -3.854789594187423 -3.825616986642957
intensification 72 -3.862699505950367 -3.8624012179891585 -3.8421009411647944
intensification 73 -3.862699505950367 -3.8624012179891585 -3.8522374331108793
intensification 74 -3.862699505950367 -3.862426829594594 -3.853901038801508
intensification 75 -3.862699505950367 -3.862426829594594 -3.855935222915714
intensification 76 -3.862699505950367 -3.862426829594594 -3.8579319268060313
intensification 77 -3.862699505950367 -3.862695507878561 -3.8588801833588864
intensification 78 -3.862699505950367 -3.862695507878561 -3.8618585295925767
intensification 79 -3.862699505950367 -3.8626955079982896 -3.8624012182202456
intensification 80 -3.862699505950367 -3.8626989162510297 -3.8624116349815667
diversification 81 -3.862699505950367 -3.8626989162510297 -3.862565034408558
diversification 82 -3.8627306636397982 -3.8627306636397982 -3.862422815751877
diversification 83 -3.8627649490556006 -3.8627649490556006 -3.8620364805954903
diversification 84 -3.8627649490556006 -3.86269190689199 -3.860129343551247
diversification 85 -3.8627649490556006 -3.8626985386797923 -3.8490639443535253
diversification 86 -3.8627649490556006 -3.8627183289377705 -3.855597512458891
diversification 87 -3.8627649490556006 -3.8627270784644567 -3.853286805919417
diversification 88 -3.8627649490556006 -3.8627053991826212 -3.853250967801506
diversification 89 -3.8627649490556006 -3.8625141304263804 -3.8573233321704357
intensification 90 -3.8627649490556006 -3.8626112016477707 -3.844817239845896
intensification 91 -3.8627649490556006 -3.862705511038773 -3.855562971521127
intensification 92 -3.8627649490556006 -3.862705511038773 -3.858313972477987
intensification 93 -3.8627649490556006 -3.862707090490137 -3.859911966951706
intensification 94 -3.8627649490556006 -3.862707090490137 -3.8613230917678703
intensification 95 -3.8627649490556006 -3.8627265112838076 -3.862524862543828
intensification 96 -3.8627649490556006 -3.862762650483876 -3.862589431711628
intensification 97 -3.8627649490556006 -3.862762650483876 -3.8626635741852753
intensification 98 -3.8627649490556006 -3.8627626545433356 -3.86268816800021
Meta-mutation
diversification 99 -3.8627649490556006 -3.8627626545433356 -3.778198386001907
diversification 100 -3.8627649490556006 -3.8587081850965403 -3.8250086628183455
diversification 101 -3.8627649490556006 -3.8594179285830275 -3.833247606266683
diversification 102 -3.8627649490556006 -3.8608353998088454 -3.8276344441123653
diversification 103 -3.8627649490556006 -3.860996804037 75 -3.830461823949748
diversification 104 -3.8627649490556006 -3.860997619430284 -3.833906005414652
diversification 105 -3.8627649490556006 -3.8572116102441125 -3.829320815976491
diversification 106 -3.8627649490556006 -3.846874624862612 -3.821053115327693
diversification 107 -3.8627649490556006 -3.848549173981322 -3.8203379602341694
intensification 108 -3.8627649490556006 -3.8415362358698615 -3.824086479952527
intensification 109 -3.8627649490556006 -3.8483291986014305 -3.8367815445872537
intensification 110 -3.8627649490556006 -3.8483291986014305 -3.840057852694451
intensification 111 -3.8627649490556006 -3.8483291986199686 -3.841878742042075
intensification 112 -3.8627649490556006 -3.8487784484161622 -3.843751257435822
intensification 113 -3.8627649490556006 -3.849688390084099 -3.8456988599689748
intensification 114 -3.8627649490556006 -3.8513273157195096 -3.848180468918093
intensification 115 -3.8627649490556006 -3.8513273157198595 -3.8486197579622066
intensification 116 -3.8627649490556006 -3.852244424684365 -3.8489781527572786
diversification 117 -3.8627649490556006 -3.852244424684365 -3.849688390084099
diversification 118 -3.8627649490556006 -3.852486081956358 -3.8490925214206415
diversification 119 -3.8627649490556006 -3.8517082370428923 -3.846501249308255
diversification 120 -3.8627649490556006 -3.854601329338985 -3.8457877669619642
diversification 121 -3.8627649490556006 -3.8546030910385563 -3.842167417156847
diversification 122 -3.8627649490556006 -3.8542921720417036 -3.8454565576323008
diversification 123 -3.8627649490556006 -3.852346281157936 -3.8451439903186317
diversification 124 -3.8627649490556006 -3.853909826103859 -3.8424039721770695
diversification 125 -3.8627649490556006 -3.853911456421052 -3.8191829202674556
intensification 126 -3.8627649490556006 -3.850935846308492 -3.8285485592013577
intensification 127 -3.8627649490556006 -3.855209103399663 -3.847002082696562
intensification 128 -3.8627649490556006 -3.8555417223896202 -3.8481976348962914
intensification 129 -3.8627649490556006 -3.855541722389936 -3.850842204000201
intensification 130 -3.8627649490556006 -3.855542215231163 -3.8516245831125473
intensification 131 -3.8627649490556006 -3.855542215231163 -3.8529619300268747
intensification 132 -3.8627649490556006 -3.856146266776749 -3.85494860111595
intensification 133 -3.8627649490556006 -3.8562062543551843 -3.8554268218168493
intensification 134 -3.8627649490556006 -3.8566077249125925 -3.855625132098489
diversification 135 -3.8627649490556006 -3.8568500193009054 -3.855880538112715
diversification 136 -3.8627649490556006 -3.8568500193017385 -3.855625126820425
diversification 137 -3.8627649490556006 -3.8566411822306517 -3.8536048917475982
```

diversification 138 -3.8627649490556006 -3.8575175178758117 -3.851650155960237
diversification 139 -3.8627649490556006 -3.856804561809514 -3.8373137176304732
diversification 140 -3.8627649490556006 -3.8579422103970473 -3.8352113089112567
diversification 141 -3.8627649490556006 -3.8583640181176944 -3.850712143387714
diversification 142 -3.8627649490556006 -3.8559517187419754 -3.8428522391158584
diversification 143 -3.8627649490556006 -3.8574392461409444 -3.831472499453096
intensification 144 -3.8627649490556006 -3.8590835480182326 -3.8329565855143315
intensification 145 -3.8627649490556006 -3.8590914820876416 -3.8511372654838887
intensification 146 -3.8627649490556006 -3.8590914820876416 -3.855073158807381
intensification 147 -3.8627649490556006 -3.860398928835998 -3.856621164047267
intensification 148 -3.8627649490556006 -3.8603991783287626 -3.8586019281033024
intensification 149 -3.8627649490556006 -3.861635547550804 -3.8590836575732235
intensification 150 -3.8627649490556006 -3.861635547550804 -3.8591197726983397
intensification 151 -3.8627649490556006 -3.8616355475508133 -3.860399270122861
intensification 152 -3.8627649490556006 -3.861674679782309 -3.860424753490347
diversification 153 -3.8627649490556006 -3.861830292081393 -3.8610824372664982
diversification 154 -3.8627649490556006 -3.8616747209646514 -3.860426920569499
diversification 155 -3.8627649490556006 -3.8615449384842644 -3.8565144800969104
diversification 156 -3.8627649490556006 -3.8616974314197456 -3.8565580131180193
diversification 157 -3.8627649490556006 -3.861251309652175 -3.8503171236947757
diversification 158 -3.8627649490556006 -3.860484182381296 -3.8471768360577023
diversification 159 -3.8627649490556006 -3.8615758518264034 -3.830543254651835
diversification 160 -3.8627649490556006 -3.8606321027802717 -3.8486832323039195
diversification 161 -3.8627649490556006 -3.8616502646948256 -3.8501664022917526
intensification 162 -3.8627649490556006 -3.8616865085577277 -3.8511343123081216
intensification 163 -3.8627649490556006 -3.8616865247584062 -3.858961055187735
intensification 164 -3.8627649490556006 -3.861918274599446 -3.859950929451119
intensification 165 -3.8627649490556006 -3.861918274599446 -3.860463247857864
intensification 166 -3.8627649490556006 -3.8619182769435025 -3.8608085339484493
intensification 167 -3.8627649490556006 -3.8626404273881105 -3.861428983354762
intensification 168 -3.8627649490556006 -3.8626404273881105 -3.8617241731610035
intensification 169 -3.8627649490556006 -3.8626404273881105 -3.8617743291686057
intensification 170 -3.8627649490556006 -3.8626404273881105 -3.861918274599446
diversification 171 -3.8627649490556006 -3.8626404273881105 -3.861998500873717
diversification 172 -3.8627649490556006 -3.862663322351086 -3.8619381779682582
diversification 173 -3.8627649490556006 -3.8626298130471364 -3.857425227171921
diversification 174 -3.8627649490556006 -3.8624286729889468 -3.853534148295095
diversification 175 -3.8627649490556006 -3.8627609621207415 -3.85446639080233
diversification 176 -3.8627649490556006 -3.862243620615494 -3.853918364537944
diversification 177 -3.8627649490556006 -3.860669797834591 -3.8399853817443557
diversification 178 -3.8627649490556006 -3.861943207929431 -3.8383235056799734
diversification 179 -3.8627649490556006 -3.8619432079294316 -3.833477584005102
intensification 180 -3.8627649490556006 -3.8586189985994195 -3.8326652553177585
intensification 181 -3.8627649490556006 -3.8622076698516534 -3.850294033203766
intensification 182 -3.8627649490556006 -3.86220767079147 -3.8550171491188285
intensification 183 -3.8627649490556006 -3.86220767079147 -3.8586189985994195
intensification 184 -3.8627649490556006 -3.862246542365326 -3.860333560483872
intensification 185 -3.8627649490556006 -3.862715676608429 -3.861429838757764
intensification 186 -3.8627649490556006 -3.862715676608429 -3.861728916551624
intensification 187 -3.8627649490556006 -3.862715676608429 -3.8619518882383184
intensification 188 -3.8627649490556006 -3.862715676608429 -3.8622076698516534
diversification 189 -3.8627649490556006 -3.862715676608429 -3.8622076707914705
diversification 190 -3.8627649490556006 -3.862722176367945 -3.86220767061313
diversification 191 -3.8627649490556006 -3.8627221765315194 -3.8604150398021635
diversification 192 -3.8627649490556006 -3.862720118275616 -3.8598303604147
diversification 193 -3.8627649490556006 -3.8626540332472254 -3.857489934123318
diversification 194 -3.8627649490556006 -3.8625106985216067 -3.846686143030864
diversification 195 -3.8627649490556006 -3.86261221720728 -3.8505288736692695
diversification 196 -3.8627649490556006 -3.862645735426837 3 -3.8539691071842768
diversification 197 -3.8627649490556006 -3.8623214619683894 -3.85150449302364
intensification 198 -3.8627649490556006 -3.8621464911217873 -3.8482858778066134
intensification 199 -3.8627649490556006 -3.8621464911217873 -3.8547043677906614
intensification 200 -3.8627649490556006 -3.8621464911217873 -3.8598891181745323
intensification 201 -3.8627649490556006 -3.8621464911217873 -3.861130382405082
intensification 202 -3.8627649490556006 -3.862146491293218 -3.8615427982229362
intensification 203 -3.8627649490556006 -3.8623998657728085 -3.861632849519676
intensification 204 -3.8627649490556006 -3.8623998657728085 -3.8617583660938273
intensification 205 -3.8627649490556006 -3.862566241423230 5 -3.8621790826815423
intensification 206 -3.8627649490556006 -3.8625662414232305 -3.862263183479744
diversification 207 -3.8627649490556006 -3.8625662414232305 -3.862320993508626
diversification 208 -3.8627649490556006 -3.8624656445063525 -3.8622515566405533
diversification 209 -3.8627649490556006 -3.8623873705166196 -3.8613616122918044
diversification 210 -3.8627649490556006 -3.862534663122469 -3.8606019361153354
diversification 211 -3.8627649490556006 -3.862534663122469 -3.8453257382492994
diversification 212 -3.8627649490556006 -3.862631448360035 -3.8537467349496324
diversification 213 -3.8627649490556006 -3.862243944320033 -3.852170230333846
diversification 214 -3.8627649490556006 -3.862321295803061 -3.8531709017979274
diversification 215 -3.8627649490556006 -3.8625540681183 02 -3.853125014215401
intensification 216 -3.8627649490556006 -3.862666886038779 -3.8567842203480387
intensification 217 -3.8627649490556006 -3.8626832594369 41 -3.8600239267868313
intensification 218 -3.8627649490556006 -3.8626832594516 17 -3.862065005713279
intensification 219 -3.8627649490556006 -3.8626832594516 17 -3.862357791386031
intensification 220 -3.8627649490556006 -3.8626832594516 17 -3.862455073769196
intensification 221 -3.8627649490556006 -3.8626832594516 17 -3.862536521658029
intensification 222 -3.8627649490556006 -3.8627127658335327 -3.862610486726722
intensification 223 -3.8627649490556006 -3.8627319837339384 -3.862648061082176
intensification 224 -3.8627649490556006 -3.8627409253781106 -3.8626668860799613

```
diversification 225 -3.8627649490556006 -3.8627409253781106 -3.8626813877068913
diversification 226 -3.8627649490556006 -3.862729839260610 -3.862666886050242
diversification 227 -3.8627649490556006 -3.86272788874159 -3.861131469008799
diversification 228 -3.8627649490556006 -3.86272788874159 -3.85101538213551
diversification 229 -3.8627649490556006 -3.862506877407904 -3.853683904202377
diversification 230 -3.8627649490556006 -3.8627451636096013 -3.8404831152425265
diversification 231 -3.8627649490556006 -3.8627067288476327 -3.849439921721639
diversification 232 -3.8627649490556006 -3.862101829035096 -3.8519008922446805
diversification 233 -3.8627649490556006 -3.86270563673386 52 -3.842309559877364
intensification 234 -3.8627649490556006 -3.862723971535523 -3.834718397390105
intensification 235 -3.8627649490556006 -3.862723971535523 -3.853802934507303
intensification 236 -3.8627649490556006 -3.8627239715446273 -3.86115982669605
intensification 237 -3.8627649490556006 -3.8627364636633033 -3.8621588217480065
intensification 238 -3.8627649490556006 -3.8627364636633033 -3.862464101582196
intensification 239 -3.862767025331113 -3.862767025331113 -3.8625818202936495
intensification 240 -3.8627732654328635 -3.8627732654328635 -3.8627081298458448
Meta-mutation
intensification 241 -3.8627732654328635 -3.8627732654328635 -3.781032626195846
intensification 242 -3.8627732654328635 -3.8627732654328635 -3.8435474790143545
diversification 243 -3.8627732654328635 -3.8627732654328635 -3.852544554479699
diversification 244 -3.8627732654328635 -3.861292324461449 -3.8545819416063236
diversification 245 -3.8627732654328635 -3.8610476955682365 -3.8470532722737865
diversification 246 -3.8627732654328635 -3.860047051803857 -3.8453796293951457
diversification 247 -3.8627732654328635 -3.8615371312755866 -3.8485061165011034
diversification 248 -3.8627732654328635 -3.8609799841887247 -3.85132557342046
diversification 249 -3.8627732654328635 -3.8610086472727314 -3.8524731984293106
diversification 250 -3.8627732654328635 -3.860974646078504 -3.841167046289962
diversification 251 -3.8627732654328635 -3.860315543692912 -3.84324612641738
intensification 252 -3.8627732654328635 -3.861642598722168 -3.842822987316158
intensification 253 -3.8627732654328635 -3.8616425987347784 -3.854303201447289
intensification 254 -3.8627732654328635 -3.8616425987347784 -3.858043278461197
intensification 255 -3.8627732654328635 -3.8626520805320963 -3.859546763488804
intensification 256 -3.8627732654328635 -3.8626520805320963 -3.8599393569414358
intensification 257 -3.8627732654328635 -3.8626520805320963 -3.86144741329037
intensification 258 -3.8627732654328635 -3.862652080532668 -3.861653684193627
intensification 259 -3.8627732654328635 -3.8626746480257332 -3.8618906288595625
intensification 260 -3.8627732654328635 -3.862705924228458 -3.861948985925701
diversification 261 -3.8627732654328635 -3.862705924228458 -3.862360138594953
diversification 262 -3.8627732654328635 -3.862660818389763 -3.8618531735539054
diversification 263 -3.8627732654328635 -3.8626897134070504 -3.8618224431344776
diversification 264 -3.8627732654328635 -3.862658453980605 -3.860389941515278
diversification 265 -3.8627732654328635 -3.862540737735843 -3.856482918288128
diversification 266 -3.8627732654328635 -3.862525471737062 -3.8564681274884673
diversification 267 -3.8627732654328635 -3.86262385340038 -3.8522804326924587
diversification 268 -3.8627732654328635 -3.8625136785207226 -3.848416069014406
diversification 269 -3.8627732654328635 -3.862599277003464 -3.8449773149786255
intensification 270 -3.8627732654328635 -3.862646444750642 -3.8438508592451255
intensification 271 -3.8627732654328635 -3.862646444750642 -3.8592492347619736
intensification 272 -3.8627732654328635 -3.862646444750642 -3.8616939944407767
intensification 273 -3.8627732654328635 -3.862646444750642 -3.861929561068909
intensification 274 -3.8627732654328635 -3.862646444750642 -3.862106255207964
intensification 275 -3.8627732654328635 -3.862646444750642 -3.8624496265284005
intensification 276 -3.8627732654328635 -3.8626791348894542 -3.8625588258979935
intensification 277 -3.8627732654328635 -3.862701375959105 -3.862620352677812
intensification 278 -3.8627732654328635 -3.862701375959105 -3.8626417332704204
diversification 279 -3.8627732654328635 -3.8627561145607827 -3.8626772702300034
Meta-mutation
diversification 280 -3.8627732654328635 -3.8578412257551564 -3.6700092289588566
diversification 281 -3.8627732654328635 -3.8608671587632526 -3.843972880461969
diversification 282 -3.8627732654328635 -3.8593035806757525 -3.8468352874717224
diversification 283 -3.8627732654328635 -3.860283532686663 -3.845096168334763
diversification 284 -3.8627732654328635 -3.8616363396579096 -3.8426600064073084
diversification 285 -3.8627732654328635 -3.860241800357167 -3.840775354043296
diversification 286 -3.8627732654328635 -3.858318772892204 -3.8420792223563414
diversification 287 -3.8627732654328635 -3.860522250460611 -3.8460814280576354
intensification 288 -3.8627732654328635 -3.8597017870311268 -3.848990206602185
intensification 289 -3.8627732654328635 -3.8597017870311268 -3.8529383924185914
intensification 290 -3.8627732654328635 -3.8597017870311268 -3.8555522348311544
intensification 291 -3.8627732654328635 -3.8597017870311268 -3.8572184436267962
intensification 292 -3.8627732654328635 -3.862059574504023 -3.858033267169283
intensification 293 -3.8627732654328635 -3.8625654719472964 -3.8593670030602567
intensification 294 -3.8627732654328635 -3.8625654719472964 -3.8597017870311268
intensification 295 -3.8627732654328635 -3.8625654719472964 -3.8602256030710342
intensification 296 -3.8627732654328635 -3.862565471947609 -3.86135812195858
diversification 297 -3.8627732654328635 -3.862573230740878 -3.862059574504023
diversification 298 -3.8627732654328635 -3.862715977634627 -3.862021412555435
diversification 299 -3.8627732654328635 -3.862715724665037 -3.8611163638903716
diversification 300 -3.8627732654328635 -3.8625427921905673 -3.8523506731424626
diversification 301 -3.8627732654328635 -3.8626645569142024 -3.8508527426353605
diversification 302 -3.8627732654328635 -3.8627333158847366 -3.8403249210301458
diversification 303 -3.8627732654328635 -3.8627386677185838 -3.839117872855096
diversification 304 -3.8627732654328635 -3.8626253071225563 -3.8401658162699084
diversification 305 -3.8627732654328635 -3.862503122609448 -3.847934311597886
intensification 306 -3.8627732654328635 -3.8616436750848258 -3.83619321133194
intensification 307 -3.8627732654328635 -3.8617048741623723 -3.8501258083719585
intensification 308 -3.8627732654328635 -3.86205942822681 -3.8602931729375936
intensification 309 -3.8627732654328635 -3.8625995594213296 -3.861116393259123
```

intensification 310 -3.8627732654328635 -3.8626222295602597 -3.8616436750848258
intensification 311 -3.8627732654328635 -3.8626222295602597 -3.8620238562429368
intensification 312 -3.8627732654328635 -3.862630697368122 -3.8621947975472617
intensification 313 -3.8627732654328635 -3.862630697368122 -3.862401002231243
intensification 314 -3.8627732654328635 -3.8627230693985726 -3.862536760197365
diversification 315 -3.8627732654328635 -3.8627230693985726 -3.8625784030809562
diversification 316 -3.8627732654328635 -3.862761377877809 -3.8625181174216814
diversification 317 -3.8627732654328635 -3.862761377877822 -3.8529366103324967
diversification 318 -3.8627732654328635 -3.862761377877822 -3.860484246612233
diversification 319 -3.8627732654328635 -3.862752312325328 -3.8527920736893244
diversification 320 -3.8627732654328635 -3.8623893944404037 -3.8446246922856107
diversification 321 -3.8627732654328635 -3.8624738144027995 -3.852827292733535
diversification 322 -3.8627732654328635 -3.862263818991372 -3.8507317877744
diversification 323 -3.8627732654328635 -3.8619332247244698 -3.8502300485125014
intensification 324 -3.8627732654328635 -3.8625753048304468 -3.854165690786324
intensification 325 -3.8627732654328635 -3.8625753048304468 -3.8596136565073693
intensification 326 -3.8627732654328635 -3.8625753048304468 -3.861200978822319
intensification 327 -3.8627732654328635 -3.8626013126527985 -3.862056852927066
intensification 328 -3.8627732654328635 -3.862653873529507 -3.862200000286176
intensification 329 -3.8627732654328635 -3.8626957756991835 -3.8623956858615145
intensification 330 -3.8627732654328635 -3.8626957756991835 -3.8624820461658915
intensification 331 -3.8627732654328635 -3.8626957756991835 -3.8625033487911926
intensification 332 -3.8627732654328635 -3.86272789441194 -3.8626013126527985
diversification 333 -3.8627732654328635 -3.86272789441194 -3.862634606707353
diversification 334 -3.8627732654328635 -3.86272789441195 -3.862596224179577
diversification 335 -3.8627732654328635 -3.862661295349831 -3.859132585106321
diversification 336 -3.8627732654328635 -3.862731423926936 -3.8588202835540004
diversification 337 -3.8627732654328635 -3.862607979219858 -3.8549040280570788
diversification 338 -3.8627732654328635 -3.862616689152808 -3.8602259343371186
diversification 339 -3.8627732654328635 -3.862621825144499 -3.8603121188458664
diversification 340 -3.8627732654328635 -3.8625683006235643 -3.8574058009603913
diversification 341 -3.8627732654328635 -3.862735416681378 -3.8540681362239972
intensification 342 -3.8627732654328635 -3.86255288880323 -3.848431587356031
intensification 343 -3.8627732654328635 -3.86255288880323 -3.8575396511396405
intensification 344 -3.8627732654328635 -3.86255288880323 -3.861867604012487
intensification 345 -3.8627732654328635 -3.86255288880323 -3.862146748214489
intensification 346 -3.8627732654328635 -3.8626018561113544 -3.8623067256787933
intensification 347 -3.8627732654328635 -3.8626536466090644 -3.8624210140369235
intensification 348 -3.8627732654328635 -3.8627299614058392 -3.8624973786215047
intensification 349 -3.8627732654328635 -3.8627299614058392 -3.862533720163433
intensification 350 -3.8627732654328635 -3.8627378491781483 -3.86257696966569
diversification 351 -3.8627732654328635 -3.8627378491781483 -3.862615545524732
diversification 352 -3.8627732654328635 -3.8627299614058392 -3.8625550696074553
diversification 353 -3.8627732654328635 -3.8627140587914184 -3.8553547833814514
diversification 354 -3.8627732654328635 -3.862649777990826 -3.8404438089034105
diversification 355 -3.8627732654328635 -3.862508375538929 -3.8433213370279677
diversification 356 -3.8627732654328635 -3.8622415003606174 -3.848946524312233
diversification 357 -3.8627732654328635 -3.8612934382377406 -3.832443997544317
diversification 358 -3.8627732654328635 -3.860762488112802 -3.8263862999414258
diversification 359 -3.8627732654328635 -3.861561238994229 -3.835563934386332
intensification 360 -3.8627732654328635 -3.860518250813348 -3.841986692584116
intensification 361 -3.8627732654328635 -3.860518250813361 -3.8546922978398457
intensification 362 -3.8627732654328635 -3.861136710544767 -3.8577587263932562
intensification 363 -3.8627732654328635 -3.861136710544767 -3.8595251353907303
intensification 364 -3.8627732654328635 -3.861986069013537 -3.860518250813348
intensification 365 -3.8627732654328635 -3.861986069013537 -3.860769661910121
intensification 366 -3.8627732654328635 -3.862323515540633 -3.8612678910388776
intensification 367 -3.8627732654328635 -3.862387333130604 -3.861972204801399
intensification 368 -3.8627732654328635 -3.8627152574495067 -3.8620944320154726
diversification 369 -3.8627732654328635 -3.8627152574495067 -3.862268423294374
diversification 370 -3.8627732654328635 -3.8625677610089237 -3.862087215257533
diversification 371 -3.8627732654328635 -3.862561936902693 -3.8492167502835817
diversification 372 -3.8627732654328635 -3.8626664453172266 -3.855613475988918
diversification 373 -3.8627732654328635 -3.862683501703142 -3.8530954619124835
diversification 374 -3.8627732654328635 -3.8624072617557097 -3.8469902554400215
diversification 375 -3.8627732654328635 -3.8624072617557097 -3.8469233314142777
diversification 376 -3.8627732654328635 -3.862411746572832 -3.8494681319521784
diversification 377 -3.8627732654328635 -3.8625852364545454 -3.849344486089001
intensification 378 -3.8627732654328635 -3.862553672313437 -3.856033829292233
intensification 379 -3.862775711978461 -3.862775711978461 -3.860705254669844
intensification 380 -3.862775711978461 -3.862775711978461 -3.8616845499887194
intensification 381 -3.862775711978461 -3.862775711978461 -3.862223559624892
intensification 382 -3.862775711978461 -3.862775711978461 -3.86253304604472
intensification 383 -3.862775711978461 -3.862775711978461 -3.862543097056002
intensification 384 -3.862775711978461 -3.862775711978461 -3.862606573780833
intensification 385 -3.862775711978461 -3.862775711978461 -3.8626577943501537
intensification 386 -3.862775711978461 -3.862775711978461 -3.862694039889467
diversification 387 -3.8627773705217203 -3.8627773705217203 -3.862726531270345
Meta-mutation
diversification 388 -3.8627773705217203 -3.8072647970002285 -3.574636166130348
diversification 389 -3.8627773705217203 -3.8598156709909333 -3.7871240362150456
diversification 390 -3.8627773705217203 -3.8578083522021146 -3.8199529944078536
diversification 391 -3.8627773705217203 -3.857151118554522 -3.830042954937978
diversification 392 -3.8627773705217203 -3.8583205736826547 -3.833366053596279
diversification 393 -3.8627773705217203 -3.857910416715074 -3.8427620304905914
diversification 394 -3.8627773705217203 -3.8579631491495103 -3.8403958263439266
diversification 395 -3.8627773705217203 -3.859913301290792 -3.826909145324955

```
intensification 396 -3.8627773705217203 -3.8618427018952106 -3.8186012582982762
intensification 397 -3.8627773705217203 -3.8618427018952106 -3.84687421474830 4
 -3.8627773705217203 4 3
 -3.857045380162298 4 3
 -3.8568992708262915 4 3
 -3.856262958608811 4 3
inde her !
 -3.8627773705217203 4
 -3.857045380162298 4
 -3.8568992708262915 4
 -3.856262958608811 4
Counter = 0
accept reflected point
 -3.8627773705217203 4
 -3.861016155802022 4
 -3.857045380162298 4
 -3.8568992708262915 4
Counter = 1
accept reflected point
 -3.8627773705217203 4
 -3.8627010255198133 4
 -3.861016155802022 4
 -3.857045380162298 4
Counter = 2
Accept contracted point on the outside
 -3.8627773705217203 4
 -3.8627010255198133 4
 -3.8614632703635894 4
 -3.861016155802022 4
Counter = 3
Accept contracted point on the inside
 -3.8627773705217203 4
 -3.8627010255198133 4
 -3.8623378849082375 4
 -3.8614632703635894 4
Counter = 4
Accept contracted point on the outside
 -3.8627773705217203 4
 -3.8627010255198133 4
 -3.8625917210985934 4
 -3.8623378849082375 4
Counter = 5
Accept contracted point on the inside
 -3.8627773705217203 4
 -3.8627010255198133 4
 -3.8626743855919083 4
 -3.8625917210985934 4
Counter = 6
Accept contracted point on the inside
 -3.8627773705217203 4
 -3.8627414162055755 4
 -3.8627010255198133 4
 -3.8626743855919083 4
Counter = 7
Accept contracted point on the inside
 -3.8627773705217203 4
 -3.8627560128175333 4
 -3.8627414162055755 4
 -3.8627010255198133 4
Counter = 8
Accept contracted point on the inside
 -3.8627773705217203 4
 -3.8627683079425417 4
 -3.8627560128175333 4
 -3.8627414162055755 4
Counter = 9
Accept contracted point on the inside
 -3.8627773705217203 4
 -3.862769251738787 4
 -3.8627683079425417 4
 -3.8627560128175333 4
Counter = 10
Accept contracted point on the outside
 -3.8627773705217203 4
 -3.8627725256466596 4
 -3.862769251738787 4
 -3.8627683079425417 4
Counter = 11
Accept contracted point on the inside
 -3.8627773705217203 4
 -3.8627757909887483 4
 -3.8627725256466596 4
 -3.862769251738787 4
Counter = 12
Accept contracted point on the inside
 -3.8627773705217203 4
 -3.862776428683439 4
```

```
 -3.8627757909887483 4
 -3.8627725256466596 4
Counter = 13
Accept contracted point on the outside
 -3.8627773705217203 4
 -3.862776789518642 4
 -3.862776428683439 4
 -3.8627757909887483 4
Counter = 14
Accept contracted point on the inside
 -3.8627773705217203 4
 -3.862777235221715 4
 -3.862776789518642 4
 -3.862776428683439 4
Counter = 15
Accept contracted point on the inside
 -3.8627773705217203 4
 -3.862777235221715 4
 -3.8627771905114896 4
 -3.862776789518642 4
Counter = 16
Accept contracted point on the inside
 -3.8627773705217203 4
 -3.8627772799314104 4
 -3.862777235221715 4
 -3.8627771905114896 4
Counter = 17
Accept contracted point on the inside
 -3.862777378484978 4
 -3.8627773705217203 4
 -3.8627772799314104 4
 -3.862777235221715 4
Counter = 18
Accept contracted point on the inside
 -3.862777382657238 4
 -3.862777378484978 4
 -3.8627773705217203 4
 -3.8627772799314104 4
Counter = 19
Accept contracted point on the inside
 -3.8627773870883626 4
 -3.862777382657238 4
 -3.862777378484978 4
 -3.8627773705217203 4
Counter = 20
Accept contracted point on the inside
 -3.8627774056427855 4
 -3.8627773870883626 4
 -3.862777382657238 4
 -3.862777378484978 4
Counter = 21
Accept contracted point on the inside
 -3.8627774056427855 4
 -3.8627774053483765 4
 -3.8627773870883626 4
 -3.862777382657238 4
Counter = 22
Accept contracted point on the inside
 -3.8627774056427855 4
 -3.862777405572773 4
 -3.8627774053483765 4
 -3.8627773870883626 4
Counter = 23
Accept contracted point on the inside
 -3.8627774056427855 4
 -3.862777405572773 4
 -3.8627774053483765 4
 -3.862777405264587 4
Counter = 24
Accept contracted point on the inside
 -3.8627774094013017 4
 -3.8627774056427855 4
 -3.862777405572773 4
 -3.8627774053483765 4
Counter = 25
Accept contracted point on the inside
 -3.8627774100204664 4
 -3.8627774094013017 4
 -3.8627774056427855 4
 -3.862777405572773 4
Counter = 26
accept reflected point
 -3.8627774100204664 4
 -3.8627774094013017 4
 -3.8627774067280525 4
 -3.8627774056427855 4
Counter = 27
```

```
Accept contracted point on the inside
 -3.8627774100204664 4
 -3.862777409783273 4
 -3.8627774094013017 4
 -3.8627774067280525 4
Counter = 28
Accept contracted point on the outside
 -3.8627774100204664 4
 -3.862777409783273 4
 -3.8627774095253313 4
 -3.8627774094013017 4
Counter = 29
accept reflected point
R-3.8627774100204664
```