NTNU
Norwegian University of
Science and Technology

Reservoir Production Optimization Using Genetic Algorithms and Artificial Neural Networks

Mats Grønning Andersen

Master of Science in Informatics
Submission date: July 2009
Supervisor: Keith Downing, IDI
Co-supervisor: John Petter Jensen, StatoilHydro

Norwegian University of Science and Technology
Department of Computer and Information Science

Abstract

This master's thesis has investigated how methods from artificial intelligence (AI) can be used to perform and augment production optimization of subsea oil reservoirs. The methods involved in this work are genetic algorithms (GAs) and artificial neural networks (ANNs). Different optimization schemes were developed by the author to perform production optimization on oil reservoir simulator models. The optimization involves finding good input parameter values for certain properties of the model, relating to how the wells in the oil reservoir operate. The research involves straightforward optimization using GAs, model approximations using ANNs, and also more advanced schemes using these methods together with other available technology to perform and augment reservoir optimization. With this work, the author has attempted to make a genuine contribution to all the research areas this master's thesis has touched upon, ranging from computer science and AI to process and petroleum engineering.

The methods and approaches developed through this research were compared to the performance of each other and also to other approaches and methods used on the same challenges. The comparison found some of the developed optimization schemes to be very successful, while others were found to be less appropriate for solving the problem at hand. Some of the less successful approaches still showed considerable promise for simpler problems, leading the author to conclude that the developed schemes are suited for solving optimization problems in the petroleum industry.

Preface

This dissertation has been written as the final part of the author's masters degree on artificial intelligence at the Department of Computer and Information Science at the Norwegian University of Science and Technology (NTNU).

During a summer working for StatoilHydro ASA as part of their Summer Project 2008 the author got in contact with personnel working with production optimization in StatoilHydro. The task of this group is to perform predictions for oil reservoir properties and behavior. This group could provide an interesting problem for a master's thesis. After some discussion a common understanding of problems and methods were established, and NTNU supervisor Keith Downing agreed with the author that the problems this group worked with were suitable for solving with artificial intelligence (AI)-methods. StatoilHydro does not have any department where AI research is performed, and to the author's knowledge, nowhere is it used in development or optimization apart from as part of Master- and PhD theses. This was seen by StatoilHydro as an opportunity to learn about AI-methods and the usability for these methods on their problems. For the author, performing this master thesis on these problems presented an excellent opportunity to use the knowledge and methods learned from different courses and experience on a relevant real-world problem.

After initial meetings between StatoilHydro employees, NTNU supervisor Keith Downing, and the author, a StatoilHydro supervisor was found, John Petter Jensen, and a problem definition formed. The goal of this master's thesis was from the beginning to explore the use of genetic algorithms (GAs) and artificial neural networks (ANNs) on problems regarding optimization of production in underwater oil reservoir models. As part of the work the author gained access to StatoilHydro simulating tools for reservoir properties and behavior. The AI-methods were to be compared to currently used methods and the possible gains and problems determined and documented.

While exploring ways to do comparison of the performance of AI-methods to other relevant methods used for optimization, the author was encouraged to contact a PhD-student, Masoud Asadollahi, at the International Research Institute of Stavanger (IRIS). Mr. Asadollahi could provide both a relevant case used for educational and experimental purposes as well as performance data for other algorithms on this specific case.

Acknowledgements

The author would like to thank supervisors Keith Downing at NTNU and John Petter Jensen at StatoilHydro for their advice and assistance while working on this thesis. Thanks go to Masoud Asadollahi and Renato Markovinovic at IRIS for help with the Brugge case and for sharing results. Thanks also go to Mathias Bellout and Jørgen Borgesen at NTNU for providing assistance on obtaining software and introduction material to the petroleum domain in the early stages of work with this master thesis.

My supportive family and girlfriend have helped and motivated me through months of work. I am forever grateful for being blessed with the company of such wonderful people.

Contents

List of acronyms used

AI Artificial Intelligence

GA Genetic Algoritm

ANN Artificial Neural Network

IRIS International Research Institute of Stavanger

EC Evolutionary Computation

MPC Model Predictive Control

SQP Sequential Quadratic Programming

NPV Net Present Value

Chapter 1

Introduction

1.1 Oil reservoirs and optimization

The Norwegian oil-adventure started in 1971. Since then over 4 billion Sm^3 (standard cubic meters) of oil equivalents have been produced on the Norwegian continental shelf. It is estimated that around 36% of the total available resources have been produced and sold [18]. The remaining 64% of the Norwegian oil reserves are still located in subsurface reservoirs. Research is conducted on all relevant parts of the production chain to be able to extract the remaining resources as efficiently and profitably as possible. One area of focus, which is the area this dissertation has focused on, is optimization of well properties in underwater oil reservoirs to maximize the profitability of production.

The conditions under which underwater oil reservoirs are found are often hazardous and inaccessible. It follows that the inner workings and status of oil reservoirs are hard to directly observe, and expensive to monitor. Fortunately, multiple tools for constructing models and simulating production on these models exist. Researchers and engineers spend considerable effort on optimization using the available tools for approximating reservoir properties and optimizing production. Numerous methods and approaches are available, all with different advantages and disadvantages. Simulators used to recreate conditions inside an oil reservoir are often complex, and simulation often come at a high computing cost. This makes the efficiency of optimization important. By minimizing the number of simulations needed for good optimization, resources can be saved.

1.2 Artificial Intelligence

AI today offers a large array of useful tools for performing computation. Among them are genetic algorithms (GAs). GAs have been established as suitable tools for solving a large variety of problems. One application area where GAs can be seen to show their full potential is for performing optimization [8]. GAs can be used for a wide range of optimization problems, from finding the best solution for trivial scheduling problem, to producing solutions for famous challenges like the travelling salesman problem.

Another useful tool from the AI-toolbox is artificial neural networks (ANNs). This technology offers emergent functionality that allows for successful prediction of the behavior of other functions, processes, simulators or similar artifacts that produce input-output patterns [3]. This technology can be used to simplify more complex systems, and this application of ANNs is used to augment optimization in this dissertation.

1.3 Motivation

The focus of work for this master thesis has been on using methods from AI to tackle the challenges relating to production optimization of sub-sea oil reservoirs. The availability of a realistic and complex industry problem to solve with AI methods provides motivation in itself. The research is interesting as an evaluation of AI methods on a relevant problem in comparison to other methods and approaches previously used. The results are also relevant for the industry, since they provide performance comparison of different methods, and potentially powerful alternatives for use in day-to-day business.

The author has experimented with AI methods to perform or augment steps involved in production optimization of sub-sea oil reservoirs. The results of this research were then compared with the results of other methods and approaches to optimization. **The idea behind the work is to develop a system using AI methods for optimization that performs well when compared to other methods used for optimization today.**

Chapter 2 will give thorough background information about all the discussed topics and concepts. The chapter closes with a section (2.3) explaining and elaborating on more detailed goals for the thesis than has been presented in this introduction.

Chapter 2

Background

This dissertation has touched upon multiple problem domains. Methods, concepts and terminology from several areas of study other than computer science will be discussed. The purpose of the following sections is to provide introductions to all domains involved in this dissertation.

2.1 The AI field

The following subsections describe the methods used in this master thesis from the AI field of study.

2.1.1 Genetic algorithms

GAs are part of a larger group of methods in AI called evolutionary computation. These methods are inspired by processes from biology; they use concepts from natural evolution to develop solutions to problems. Genetic algorithms provide an approach to learning that is based loosely on simulated evolution [15].

2.1.1.1 Evolution in biology

In 1789, Thomas Malthus said that the growth rate of a population is a function of the population size, and therefore, if left unchecked, a population will grow exponentially. However, since environments have finite resources, the population will produce more young than the environment can sustain, and the individuals will have to compete for these resources [13]. This is where Charles Darwin entered the discussion. Darwin said that when more individuals of a species is born than can survive, a struggle for existence will occur. If an individual is born that vary slightly in any matter profitable to

itself, it will have a better chance of surviving, and thus be naturally selected [5]. In biology, evolution is observed as change in the behavior and form of organisms between generations. Properties and behaviors are inherited from the ancestors of an organism, the organism is modified in unique ways from changes in the genetic material from one generation to the next [24]. This means that there will be variance in the populations through the generations.

The result is that individuals will arise that can be significantly different from their ancestor. When there at the same time are limited resources in the environment, a struggle for these resources will ensue. The better fit individuals in this struggle will most likely persist and get to reproduce, the less fit individuals have a larger chance of disappearing. Thus, over the generations, evolution will drive the population to be better adapted to the environment. Three important concepts should be drawn from this; **selection**, **variation**, and **inheritance**. These concepts will be further explored in subsequent sections discussing their importance in genetic algorithms.

2.1.2 The inception and development of evolutionary computation

In the 1960s, advanced computing technology was becoming relatively cheap. The availability of the technology meant that computers could simulate and analyze problems more complex than what would be possible mathematically. Amongst the many groups interested in this possibility were evolutionary biologists interested in developing and testing models of natural evolutionary systems. At the same time, engineers and scientists wanted to explore the potential of using these methods which were assumed tested and proved by nature again and again to create various useful artifacts [15]. The work at this point was mostly very theoretical, with the aim of simulating natural evolution rather than harnessing the power of evolution to do computation. This changed in the 70s, when focus on early GAs was on developing more application-independent algorithms. Different subfields of evolutionary computation (EC) were unified in the 1990s, and many fundamental assumptions and underlying theories were revisited in order to strengthen and generalize the basic EC paradigms. This led to the further development and usability of the methods as competent problem solvers today [8].

2.1.2.1 The basics of genetic algorithms

Several entities that make up the building blocks of genetic algorithms have their direct counterpart in nature. Populations, individuals and their fitness,

generations and genomes are all present both in nature and in GAs. Processes like selection, reproduction, inheritance and development also have similar meaning in both systems. The author will briefly discuss each concept and explain how they together make up the genetic algorithm. Genetic algorithms are a subset of a larger family of algorithms in evolutionary computation. There are several criteria and properties that genetic algorithms have that other similar methods don't have. The author will not make a complete listing of these differences, for a comprehensive and detailed overview on genetic algorithm, please refer to Jong [8], Mitchell [15], Holland [7] or Goldberg [6].

Individuals/phenotypes These entities can be considered the most important part of the genetic algorithm. In many ways they behave like organisms found in nature. They have their own unique properties; they are part of a population of other similar individuals. They fight for survival and they reproduce. In a genetic algorithm they normally do not have complex lives like in nature, but can be considered solutions to problems. The individuals often encode a pattern or parameters for a function or simulator. These solutions can be tested and given a score [8].

Environment The individuals in a genetic algorithm exists to be tested and judged. They are judged by interacting in their environment, which is the process of calculating how good they perform at the task they are meant to be working on. The environment is often a function or simulator capable of calculating how well an individual performs [7].

Fitness This is the performance score each individual receives after interacting in its environment.

Selection This is the process of evaluating fitness of the individuals and selecting individuals that are better fit than other individuals. The selection mechanism is used when deciding parents for reproduction.

Population The population is the collection of individuals that at any point exists in the genetic algorithm.

Genotypes This is the most basic building block in a genetic algorithm. In living organisms the genotype is associated with the DNA building blocks. The DNA of the organism plays a very large part in the organisms development from inception to death. In genetic algorithms, the genotype of an

individual is the only factor determining the development of the individual. The genotype is the only trait of an individual in a GA that is not meant to be directly observable. The genotype provides the encoding for the individual. From the genotype, an individual is developed into a mature phenotype [8].

Inheritance This is the concept of how traits are transferred from parents to their offspring. In GAs all the traits transferred from parent to offspring can be found in the genotypes. The process of evolution involves change to this genetic material which will be discussed later. How well the changes in a genotype correspond to changes in the phenotype that is developed is a very large factor when determining the success of a genetic algorithm. If the mapping between genotype and mature phenotype is not very consistent and predictable, preservation of the genotypes of successful phenotypes during the process of evolution will not be reflected in the success of offspring. The profitable traits will not be inherited consistently and the performance of the genetic algorithm might become random or inconsistent [8].

Generation A generation in GAs is a point in time where one population of individuals disappears and gives room for a new population of individuals to be tested in its environment. This event occurs when all the individuals in a population have been attributed a fitness value, parents have been selected and a new pool of genotypes has been determined.

Reproduction This is the process of recombining existing genotypes into new genotypes. The genotypes of parents are used to constitute a new genotype that will be the basis for a new individual. Reproduction can be separated into sexual and asexual reproduction. In sexual reproduction cloned segments from multiple parents are used to create hybrid offspring. In asexual reproduction a single parent provides the basis for the offspring [12].

Variation The differences in genotypes are meant to be reflected in differences in phenotypes. By inducing change in the genotypes from generation to generation, the individuals in the population should exhibit different behavior and show different traits than their parents. These changes can be minor, but are essential for continued progression.

Genetic operator: Crossover Genetic operators are the forces that manipulate the genotypes and produce the changes between parent and offspring

individuals. Crossover is a genetic operator used only for sexual reproduction. The genotypes of two or more parents, typically bit strings, are cut at one point, and one part from each parent is recombined in two offspring individuals [12].

Genetic operator: Mutation This genetic operator is used in asexual reproduction, but also in addition to sexual reproduction. A part of the genome is modified. Since the genome is a string of binary digits, this is done by simply shifting one bit in the genetic code [12].

Genetic operator: Summary Much has been written regarding which genetic operator is the "better" [26]. They have advantages and disadvantages in different situations. Each represents a different heuristic that complements the other. Crossover provides exploitation of good solutions in order to produce better solutions. Mutation induces variation into the population, allowing for exploration of new solutions.

2.1.2.2 Evolution: interaction of concepts

Figure 2.1 shows how these concepts interact and form a cycle of life. When the selection mechanisms favor individuals with good fitness, this should lead to the population accumulating good traits and getting rid of bad traits, making the individuals in the population better adapted to their environment over time.

2.1.2.3 Genetic algorithms in optimization

The biological motivation for genetic algorithms has been made clear in the previous subsection.

However, the area of application for GAs that has received the most attention is optimization. There exists a large number of problems that can be represented as optimization problems, and very few available solutions for how to solve them [8]. Optimization can be observed everywhere; people aim for maximum efficiency and performance in their endeavors, physical systems in nature tend to a state of minimum energy. To make use of optimization as a tool for our problems a few important concepts are required;

Objective function value There must exist some quantitative measure of the performance of the system under study. This performance is measured by the objective function.

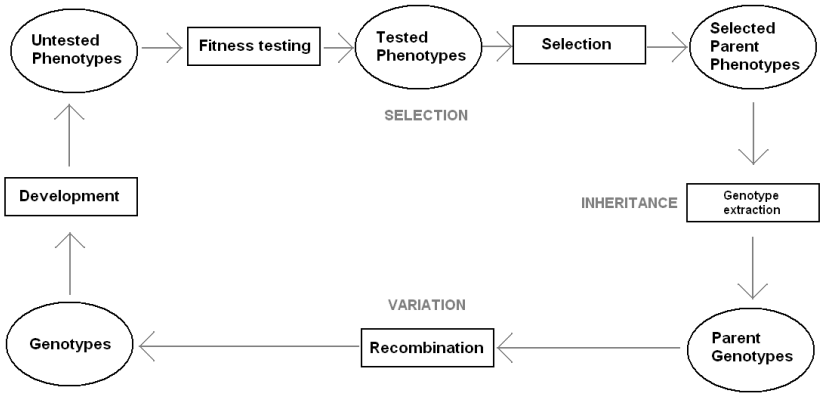

Figure 2.1: In this cycle genetic information is used to developed grown individuals, these individuals are tested for fitness, and selected for reproduction based on their fitness performance. Selected individuals transfer and recombine their genetic information to the next generation where the cycle process starts anew.

Variables The objective function value depends on characteristics of the system that can be modified to observe their impact on the objective. These are called the variables.

Optimization can be viewed as exploration of a search space [17]. Figure 2.2 shows such a search space. Variable sets are represented on the X-axis. This is often a multidimensional solution vector u, but for illustrative purposes it is represented only in one dimension. The Y-axis is the resulting performance measured by applying the variable set to the objective function f. The goal is to locate the optimal point in the search space. A point u^* is the global maximum (optimal) point if,

$$f(u^*) \geq f(u)$$

for all possible u.

For detailed information on numerical optimization, see Nocedal [17].

GAs are mathematical tools for parallel adaptive search where each individual corresponds to a sample point in a search space. The fitness measurement in GAs corresponds to the objective in general optimization, while the individuals in the population in a GA hold sets of optimization variables. The initial population is a random placement of the individuals in the search space, and through evolution the individuals located at the most fit locations in the search space are selected and recombined into new possible solutions

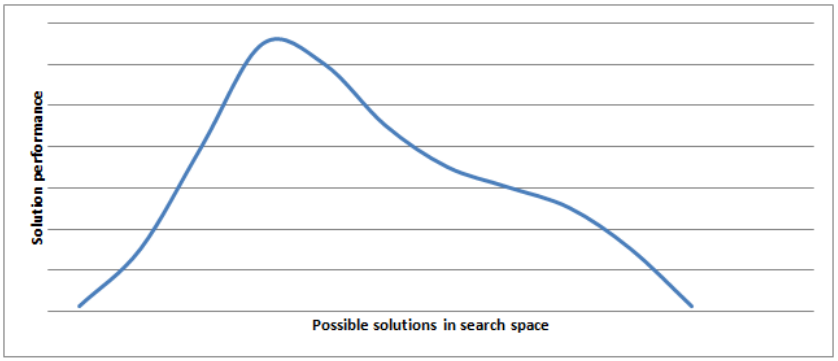

Figure 2.2: Search space optimization. The highest point on the curve corresponds to the solution in the solution space with the highest performance.

[8]. Multiple promising points in the search space can be explored simultaneously, until at some point exploitation of the best solution wins over in the selection process and the algorithm converges to one good solution. The degree of exploration provided by random initialization, the parallel search process and mutation operation acts as a guard against early convergence to local maxima. There is however no guarantee that a global maximum will be located in GAs [7].

2.1.2.4 Convergence in genetic algorithms

In nature, the environment is in a constant state of flux. The individuals in the environment have to adapt to this change over time. In nature, the purpose of evolution is to keep the population alive by adapting them to their surroundings. In practical applications the environment is often constant. This means there can exist some global maximum objective for a given problem, see section 2.1.2.3 for elaboration on optimization. In these cases, evolution can stop when this solution is identified, or a solution believed to be close to this global maximum. The challenge is in knowing when this event occurs. One solution is to keep evolving the population until no changes are made to the population that can improve it any more. The difficulty with this is that only the simplest problems converge to such a point in finite time. The principle can still be used. By measuring fitness variance in the population over the generations we can detect when evolution no longer introduces significant change to the population. While the aim is zero variance (all individuals are the optimal solution), a low value for this variance is often sufficient. Another convergence criterion that performs well is measuring the best so far fitness score. By recording this for each

generation and observing the changes in the performance, one can determine convergence when this property no longer improves. Both these measures are fairly robust and problem-independent measures of convergence[8]. Figure 2.3 shows how both these measurements should present themselves in a GA that has reached convergence. The development is usually rapid in the start and slows down some after the initial generations.

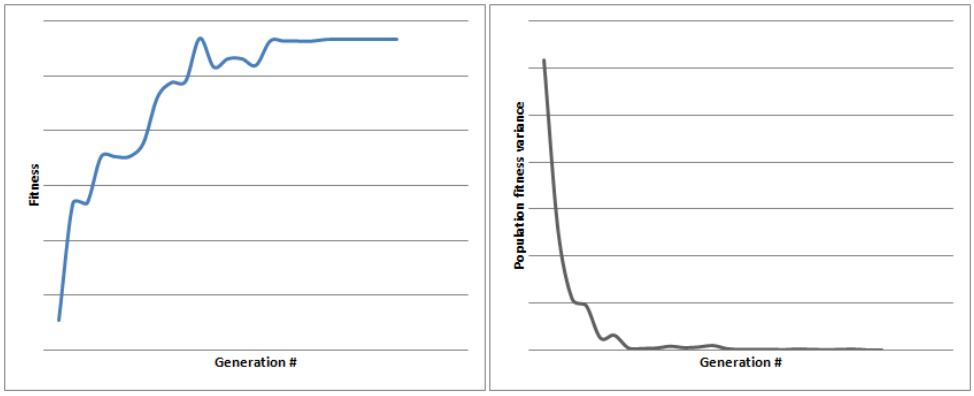

Figure 2.3: The Y-axis in (a) shows fitness for the best individual for each generation. This property is no longer improving and we can assume the system has converged to a good solution. The Y-axis in (b) shows population fitness variance for each generation. This graph indicates that the population has reached a relatively homogenous stage where all the individuals are of similar fitness.

2.1.3 Artificial neural networks

ANNs are another method in AI with roots in biology. ANNs are inspired by the neural networks constituting animal brains.

2.1.3.1 Neural networks in biology

The term "neural network" describes a population of physically interconnected neurons. Signals are passed between these neurons. This communication often involves an electrochemical process. Neurons have several connections to and from other neurons, and the net effect from all ingoing interactions in a neuron decides the behavior of the neuron, which in turn leads to the neuron sending its own response further along the network. For

a description of animal neurons and further information on biological neural networks and their properties see Bjålie [2], Kandel [9].

2.1.3.2 The basics of artificial neural networks

Neurons and their interconnections can be modeled in computers producing mathematical structures similar in functionality to their organic counterparts. These structures are called ANNs. They generally take some input, processes it and gives some output. Many variants and types of ANNs exist. The author has been using standard backpropagation ANNs for this thesis.

Structure An ANN consists of nodes (neurons) with weighted connections between them. There are some nodes that receive input, some nodes that give output, and hidden nodes in between. Each node processes all its input, for example by summing it up and running the sum through a function, and propagates its result to its connected nodes until an output is given at some output node(s) [3].

Figure 2.4 presents the topology of a simple example ANN.

Input and output Figure 2.4 displays an array of input nodes, in this case 5. On the other end, only one output node is shown. This corresponds to an input–output pattern where 5 input values produce 1 output value. The number of input and output values can vary, but this type of pattern is what ANNs typically use.

Learning Just like animal brains, ANN's learn from experience. By experiencing instances of a problem, an ANN receives information about how to update the weighted connections between neurons. This happens based on minimizing error in estimating output given desired output from a problem instance set with different inputs and outputs. This enables an ANN to adapt itself in a way that lets it generalize over the data and further enables it to make predictions about future (unknown) problems by acting as a function for the input [15]. This process is referred to as backpropagation. The implemented backpropagation algorithm is explained in detail in the methodology chapter, section 3.5.5.

Convergence Convergence can be measured by observing how the error between expected target value and actual calculated value is reduced throughout training for a set of testing data other than the training data. When the error has been reduced to a point where it is no longer decreasing this generally means the ANN has converged [3].

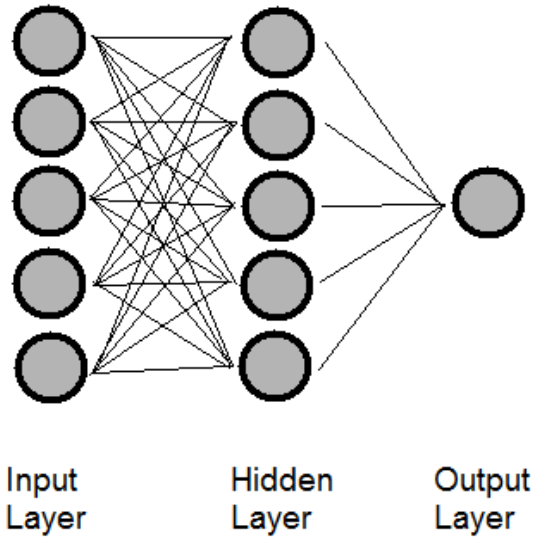

Figure 2.4: Example ANN structure. Input nodes receive input and pass it along to the hidden nodes through weighted connections. The received signal is processed in the hidden nodes and sent along weighted connections to output node(s) which further process the signal and produce the final output.

Data representation and interpretation ANNs are often built with topologies making the information at any specific node hard to interpret. The information contained in a network is often best expressed in the structure's total ability to produce correct output given an input. These steps are performed in the backpropagation algorithm [3]. ANNs trained on well defined input–output cases are capable of expressing a rich variety of decision surfaces without much direct regard to problem semantics [15]. This convergence to a state that acts as a general problem solver for a given domain is an emergent feature of ANNs. It has the ability to produce a complex mathematical structure where each single node contains distributed information processing or redundancy functionality that is very hard to interpret out of the big context [15]. ANNs are often seen as black boxes that take input and produce output, without the designer bothering too much about perfect understanding of what comes in between.

Generalization When an ANN produce correct output for the majority of input cases other than the ones in was trained with, it can be said to generalize well. A well trained ANN for the purposes of approximation like was used in this dissertation should provide a smooth nonlinear mapping. This means that it should be able to interpolate to new cases that are similar but not identical to those patterns used for training. If the network is overtrained, this will result in a non-smooth mapping, and the ANN will work more like a memory with direct lookup from input to output. The structure of the ANN is therefore an important factor for correct training. A correct amount of hidden layers and nodes should be used. If more are used than are required to learn the input-output relationship, there will be more weights than necessary which can lead to overfitting of the data and bad performance when approximating unknown cases [3].

More information on ANNs is available by Callan [3] and Mitchell [15].

2.2 Reservoir production optimization

This section will serve as an introduction to the part of this dissertation which concerns itself with the petroleum industry domain. The focus will be on providing descriptions of relevant models and problems covered by this dissertation.

2.2.1 Exploration and development of oil fields

This subsection will briefly illustrate the structure and properties of oil reservoirs to make clear the process of performing optimization on models of these.

Origin of oil and gas

Prehistoric organic material deposited at the bottom of oceans and swamps make the basis for oil and gas found today. Stacked layers of organic materials are turned to oil and gas over millions of years under high pressure and temperature. Water has a higher density than oil and gas, therefore oil and gas will migrate to the surface unless trapped inside reservoirs without possibility of escape [29].

Exploration of oil and gas

Oil and gas is generally located in very inaccessible areas (else it would have easily escaped a long time ago). Oil reservoirs can be found several kilometers beneath the surface. These reservoirs are detected by means of seismic imaging. Sound waves are fired towards the ground, and the time they take to bounce back is measured. This time will differ in different layers of rock, and these differences make it possible for geoscientists to create 3D maps of the subsurface ground. When an area is located that has the potential for containing oil and gas, exploration drills determine whether oil and gas is indeed present. The performance of these wells together with the seismic data is then used to predict how field development decisions will affect future production. A field [1] will be developed if the initial predictions are promising [29].

Development of oil and gas

In this phase, wells are drilled into the subsurface ground, and connected to surface facilities. Wells are created by drilling holes and cementing a steel pipe inside. Holes in this pipe are made to allow for hydrocarbons and water to flow in and out of the reservoir [29]. Figure 2.5 shows an example of a reservoir with several wells.

Production of oil and gas

This phase can go on for tens of years and is divided into three stages. In the initial stage the pressure inside the reservoir is high, and hydrocarbons

[1] An oil field is a collection of reservoirs related to the same geological structure [29].

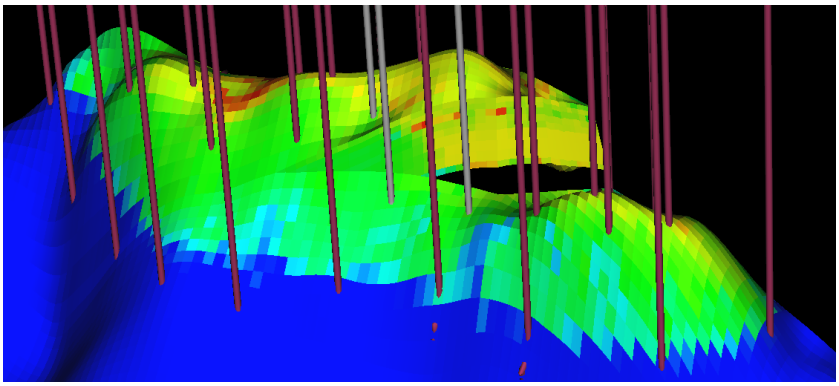

Figure 2.5: Reservoir with multiple wells. This illustration shows the Brugge case model using the ECLIPSE simulator tools. The color shows concentrations of oil and water.

are driven into the well producers. This process is referred to as primary recovery. Around 10% of the available hydrocarbons are typically extracted in this phase. In the next phase, the secondary recovery, the pressure inside the reservoir has declined as a result of primary recovery. This creates a need to inject fluids into the reservoir to increase the pressure to be able to extract the hydrocarbons. It is this phase which serves as the focus of this master's thesis. In the tertiary recovery phase chemicals that alter the properties of oil can be used to further the recovery of this oil [29].

Field development management

When the necessary structures for oil extraction is in place, production can commence and performance data is preserved. When enough field data has been recorded, numerical reservoir simulation models are developed on the background of these data. These models of the subsurface ground seek to describe the effect of changing input parameter for process control on the hydrocarbon production. These models have time-varying (dynamic) and time-invariant (static) properties. The dynamic properties are fluid pressures and saturations. Remaining fluid properties, viscosity and density, and geological properties[2] (permeability and porosity) are generally considered to be static properties [29].

The geological properties of a reservoir can vary significantly over space. These differences decide the flow paths for fluids inside the reservoir. The goal of the simulator is to determine optimal flow paths for fluids to extract

[2]For detailed information on geological properties and fluid properties, see Dake [4].

the maximum amount of oil on one side, given injection of fluids on the other. Figure 2.6 depicts this situation in a 2D reservoir. We can see the oil-water front being irregular in different spatial locations in the reservoir. By adjusting injection rates from water injectors over time, this oil-water front is adjusted to maximize production of oil in the oil producers and minimize production of water in these producers (for late phases of production when there is little oil left) [29].

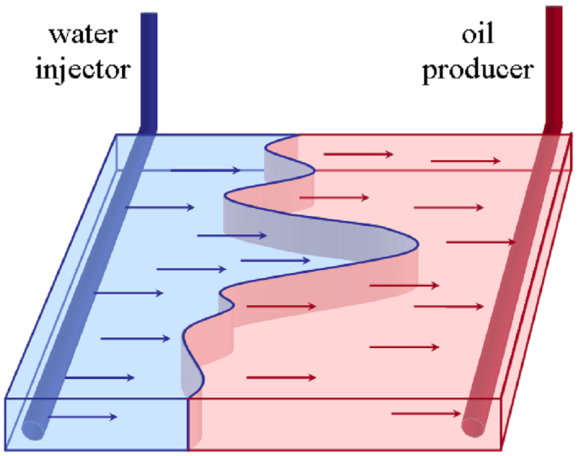

Figure 2.6: Horizontal injection and production well. This illustration is borrowed from Zandvliet [29].

2.2.2 Use of simulator models for optimization

In industry today, computers are often given the responsibility for advanced control of process plants. Advances and cost reduction in computer hardware, competitive business environments and academic interest have all played their part in this development. Model predictive control (MPC) is currently the most widely implemented advanced process control technology for process plants [22]. Academic interest in MPC was established in the 1980s, and led to a thorough understanding of its theoretical properties [20], [21]. MPC systems have been developed for use in process control under many different names for decades, but the academic approach led to a strong conceptual and practical framework for both practitioners and theoreticians [16]. As the name implies, MPC involves using models. These models represent some real-world process. MPC involves a lot more than just using models, but this falls outside the scope of this dissertation as an AI study.

The author has used models of reservoirs to simulate reservoir behavior. These models can be used to predict and analyze dependent variables (output) with respect to changes in independent variables (input). The added value of using models comes from being able to predict the outcome of a process before the fact. By being able to find optimal input parameters for a given industrial process before performing it, a lot of resources can be saved by not having to try different solutions on the actual problem. Control performance can be improved, downtime and maintenance requirements can be reduced, in short, the day-to-day production can become a lot more flexible and agile [16]. For reservoir production optimization the challenges related to observation, process control and production can be quite extreme [29]. Good prediction of behavior, properties, and performance before the fact is therefore of critical importance for the feasibility and profitability of the project.

Figure 2.7 shows how a simulator can be used for optimization. The simulator receive some initial input from an optimizer, and returns the performance results of optimization back to the optimizer. The optimizer makes some alteration to the independent input variables, sends the new input to the simulator and receives the new output. The effect of changing the input parameters can then be analyzed, and some heuristic in the optimizer will decide how the input parameters can be further changed in a good direction.

2.2.3 The ECLIPSE reservoir simulator

This subsection provides information about the oil reservoir simulator used by the author for the cases discussed in this dissertation. The ECLIPSE simulator is developed and maintained by Schlumberger Information Solutions (SIS). ECLIPSE has "been the benchmark for commercial reservoir simulation for over 25 years because of their breadth of capabilities, parallel scalability, utility computing, and unmatched platform coverage."[25].

The following description is taken from the ECLIPSE manual [25].

> The ECLIPSE simulator suite consists of two separate simulators: ECLIPSE 100 specializing in black oil modeling, and ECLIPSE 300 specializing in compositional modeling. ECLIPSE 100 is a fully-implicit, three phase, three dimensional, general purpose black oil simulator with gas condensate options. ECLIPSE 300 is a compositional simulator with cubic equation of state, pressure dependent K-value and black oil fluid treatments. ... Both pro-

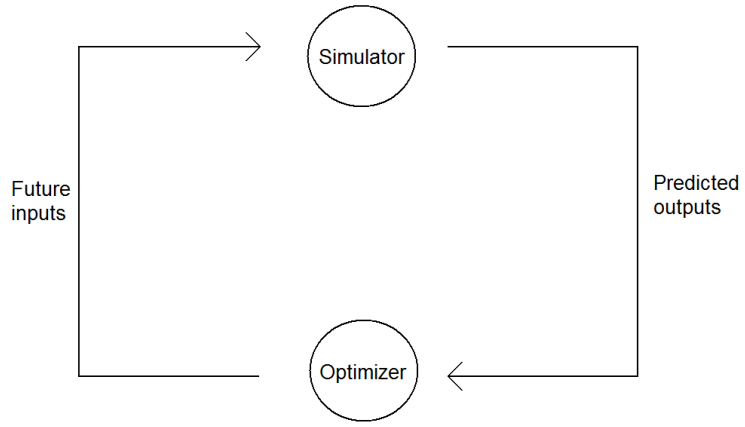

Figure 2.7: Use of a simulator for optimization. An optimizer sends input to an objective function or simulator which returns an objective function value. By experimenting with different input according to some heuristic, the optimizer can find optimal inputs for producing desired output.

> grams are written in FORTRAN and operate on any computer with an ANSI-standard FORTRAN90 compiler and with sufficient memory. For large simulations the simulators can be run in parallel mode. The Parallel option is based on a distributed memory architecture implemented using MPI (message passing interface).

3-dimensional models of the terrain of oil reservoirs are made up of grid cells with unique geological and fluidic properties as discussed in section 2.2.1. Placement of wells is specified and measurements of factors like pressure and saturation in the reservoir and wells can be performed. These models are initiated based on some history data. They are then able to make predictions on future data. A production horizon is specified in ECLIPSE models, this is the period of time the production will go on for. While it is possible to change parameters at any point in the production horizon, this will lead to optimization scenarios more complex than strictly necessary for the purposes of this dissertation. Previous studies which the author used for comparison did also not change parameter values over time. The full specifics of the ECLIPSE simulator is out of the scope of this dissertation, for more detailed information, see [25].

From the author's point of view, using ECLIPSE has proved to be easy and suitable for the purposes of this thesis. It is worth noting the similarity

between the ECLIPSE simulator and previously discussed ANNs. Both artifacts take inputs and output according to some pattern and can be used to predict future instances of problems.

2.2.4 Optimization of reservoir model production

This subsection contains information on some methods used for optimization today.

2.2.4.1 Sequential quadratic programming

Methods based on sequential quadratic programming (SQP) are considered amongst the most effective methods for nonlinearly constrained optimization. SQP discovers objective function derivates by intentionally mutating the optimization parameters and observing the effect. These derivatives are used to determine the appropriate direction of change in the optimization parameters. This is a very simplified view of SQP. For detailed information, see [17].

Optimization can be performed by using Matlab [14] scripts and tools. Matlab has a toolbox for optimization, where a lot of optimization methods are available. Amongst these is the function *fmincon*, which is Matlab's SQP-implementation. In the Matlab documentation [14], the following is written about the SQP method:

> SQP methods represent the state of the art in nonlinear programming methods.. . . , the method allows you to closely mimic Newton's method for constrained optimization. . . . the principal ideal is the formulation of a QP subproblem based on quadratic approximation of the Lagrangian function. . . .

Optimization results produced in this master thesis will be compared to results produced by optimization using Matlab's *fmincon* function. These results are from the research performed by Lorentzen et al. [11]. This research showed that optimization using SQP is robust and efficient for the case presented later in this dissertation.

2.2.4.2 Hooke-Jeeves method

This is another method that has previously been used for optimization on one of the cases this dissertation will involve. Hooke-Jeeves is a search method for finding the minimum (or maximum) of a multidimensional surface. The method is a sequential technique that at each step performs two kinds of

moves, an exploratory move and the pattern move. The exploratory move explores the behavior of the objective function when parameters are altered, and the pattern move takes advantage of the pattern direction established by the first move. These moves are repeated until convergence. For detailed information about Hooke-Jeeves, see [23].

Results of optimization by this method will also be compared to the results produced by the author. These results are from research performed by Asadollahi et al. [1].

2.2.4.3 Previous uses of GAs and ANNs in the petroleum industry

During the planning phases of this master thesis, the author searched for relevant previous studies performing optimization using AI techniques in the domain. The author has not been able to locate any identical application schemes for use in the domain, but papers have been written on using GAs for similar tasks.

Yang et al. [28] presents the most similar case to what the author is trying to accomplish. In this article, GAs are compared to simulated annealing (SA) algorithms (see [10] for details on SA) for parameter optimization on a production-injection operation systems (PIOS), consisting of injectors, reservoir, producers and surface facilities. In all, similar to what is performed in this dissertation. The research involves optimization over more of the parts involved in the production chain than what this dissertation has considered. The article does not mention any specifics on the GA used, but shows GA performance to be as good as SA for the problem task [28]. Field performance showed that the optimization techniques increased the economic benefits and extends the reservoir life.

Velez-Langs [27] presents a very good overview of other applications for GA in the petroleum industry.

In other industry domains, the use of both GA and ANN seems more common. Park et al. [19] discusses using GA and ANN for aluminum laser welding automation, this article features some detail on GA parameters used and describes a successful application of the GA as an optimizer for the problem. This article also used an ANN to judge the performance of the GA solutions, and this method was shown to be effective [19].

2.3 Goals

Section 2.2.4 discussed optimization using SQP. This method acts as an optimizer on a reservoir simulator. Use of this method does not require anything of the simulator used apart from it being able to take input and produce output. Similarly, the simulator used is not dependent on the optimization algorithm used. This modularity is illustrated in figure 2.7. This modularity implies that the different parts of the figure can be interchanged.

As described in section 2.1.2.3, GAs can be used as optimizers. The potential for use of GAs as optimizers in this domain has been the main focus in this thesis.

Goal 1 Explore the potential for using genetic algorithms as optimizers for use with reservoir simulators. The success of this approach will be compared to other means of performing optimization on the reservoir simulator, as discussed in section 2.2.4.

Section 2.2.3 describes similarities between reservoir simulators and ANNs. One goal of this master thesis is to research the possibility for training ANNs based on data from simulation and compare the performance of these ANNs with that of the original simulator.

Goal 2 Explore the potential for training ANNs based on data from simulations, and running these ANNs as simplified models of the simulators.

While these two goals focus on studying AI-methods, the whole reason for performing such studies is to develop feasible and efficient methods for use in the industry. This aspect of the problem must be taken into consideration and is expressed in the following goal:

Goal 3 From a industry perspective, the goal and point of the work in this dissertation is in exploring the potential for reducing number of simulations and saving time, while still achieving good optimization results.

Chapter 3

Methodology and implementation

This chapter explains how the different technologies described in the previous chapter were used together to perform oil reservoir production optimization using methods from the AI field of research.

3.1 Problem and possible gains

Apart from the academic interest and curiosity regarding using AI methods on reservoir optimization problems, there are challenges in the domain today that this research is trying to address. As will be discussed in section 3.2, a simulation run in a reservoir simulator can take very much time. For very big models and long production horizons, the run time for a single simulation can be several hours. When performing production optimization on a reservoir simulator, several thousand simulations can be necessary. The total time for the optimization run can be considerable.

The search space represented in the reservoir simulator is often complex and nonlinear. As discussed in section 2.1.2.3, GA as a search method is a powerful tool, and can provide a valuable alternative to other optimizers on nonlinear problems.

The potential for saving time is also the basis for using ANNs as simplified models of reservoir simulators. If a limited number of simulations can be performed on the simulator and provide a good enough training base for the ANN, the simulator can be replaced by the ANN. For complex simulations, the time to train an ANN on thousand of cases is significantly less than running a single simulation, and the time it takes to run a simulation on the ANN can be considered instant compared to a simulation run on the

model[1]. The simulation time alone for the ANN is irrelevant without taking into consideration the time it takes to generate training data and perform the training. A comparison of this method to running all simulations on the reservoir simulator must take all such factors into account.

These observations together with the modularity discussion earlier in this section leads to a number of possible schemes for using AI-methods together with the reservoir simulator and existing optimizers. Figure 3.1 illustrates how the modularity of the problem allows for combining the use of the different methods. The author has chosen 3 of these schemes as the focus of this master's thesis, and these are discussed in the following subsections.

3.1.1 Scheme 1: using GA as an optimizer for the reservoir simulator

In this scheme the GA acts as an optimizer for the reservoir simulator. It is capable of managing a population of individuals containing vectors of input parameters for the reservoir simulator. When a simulation is performed by the simulator using the input parameters, a performance score can be measured, see section 3.2. This performance score will be optimized by the GA through the process of evolution on the population of individuals. Figure 3.2 illustrates how this scheme is intended to work. A 2-dimensional figure only allows for 1-dimensional parameter vectors. The points on the X-axis should instead be considered a simplified representation of n-dimensional solution vectors. While this representation is not realistic, it is very suitable for illustrative purposes.

The GA steps found in section 3.4 summarizes scheme 1.

3.1.2 Scheme 2: using GA as an optimizer for an ANN built from data from reservoir simulations

In this scheme training data will be generated from running the reservoir simulator with varied input parameters. An ANN can then be trained from this training data. The training data should cover as much of the input parameter space as possible in as few simulations as possible. The ANN can then be used as a model for the GA to perform optimization on. The aim is to perform the optimization with fewer or a similar number of simulations

[1]Simulation time for the Brugge case was recorded to around 90 seconds, the same simulations were performed in around 0.0005 seconds in the ANN. Feedforward in the ANN is performed about 180000 times faster than simulation in the reservoir simulator.

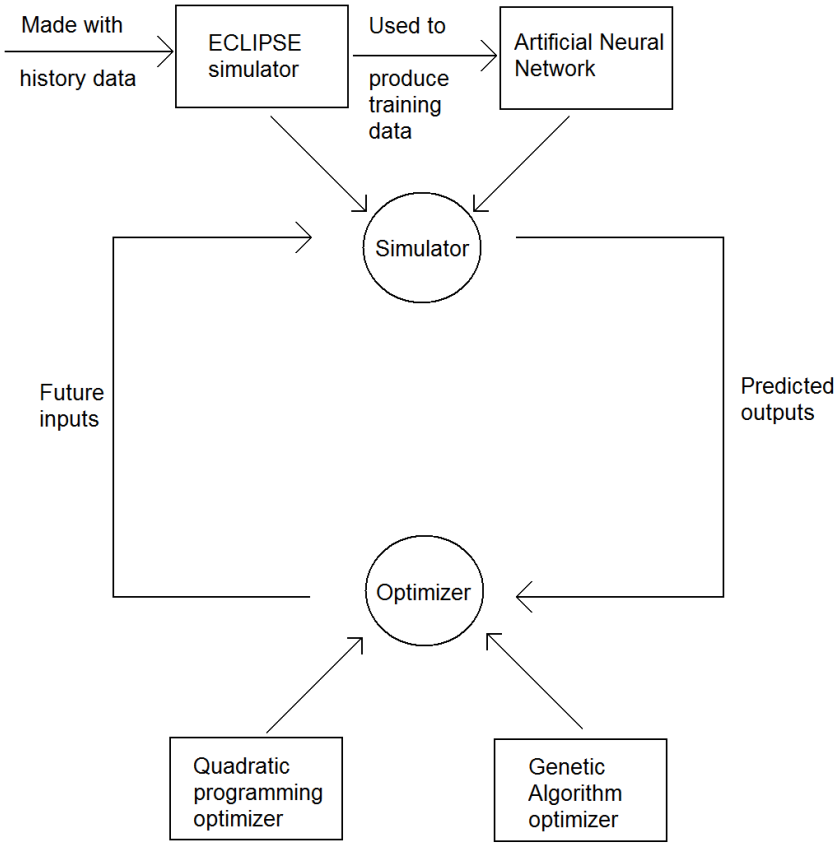

Figure 3.1: Combination of methods. The simulator and optimizer parts are modular and problem-independent. Successful solution can be developed involving different approaches.

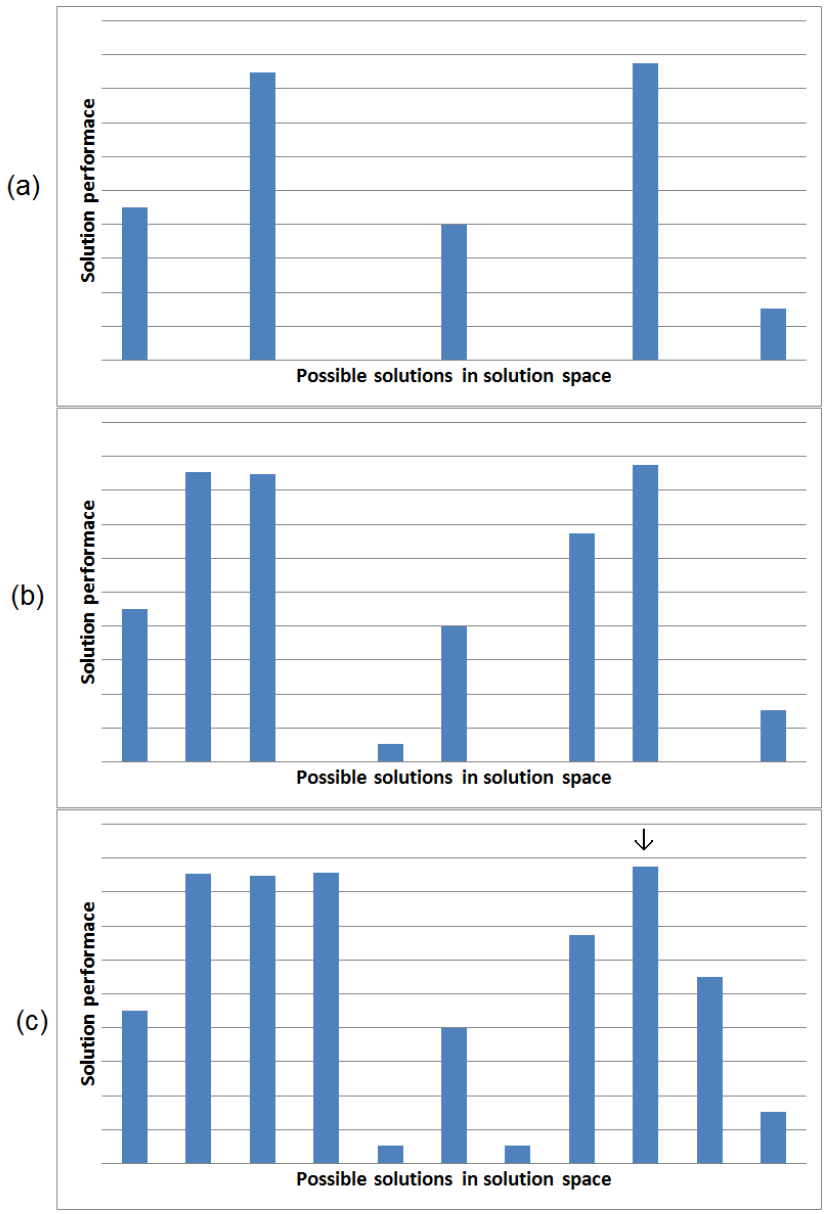

Figure 3.2: Scheme 1: (a): An initial population is scattered across the solution space. (b): Through selection, the most fit solutions will be explored and reproduced into several good solutions. (c): After a finite number of generations, the algorithm will have converged and a single best individual can be pinpointed.

than in Scheme 1. This means that the number of training cases should be limited below a certain threshold determined by scheme 1. Figure 3.3 illustrates how this scheme is intended to work.

The following steps summarizes scheme 2.

```
1 training data is generated and simulated
2 ANN is trained using the training data
3 GA-optimization is performed on the ANN
```

3.1.3 Scheme 3: using GA as an optimizer on a combination of reservoir simulation and ANN

In this scheme, the model component in figure 3.1 will be changed back and forth between the reservoir simulator and ANN several times through the GA optimization. The idea is to generate a random initial population with the GA and run these individuals on the reservoir simulator. The GA will continue to develop this population using the MPC simulator. After a sufficient amount of training data has been generated, an ANN can be trained from the data. The reservoir simulator can then be replaced by the ANN, but only for a limited number of generations. Since the time it takes to optimize using the ANN is a lot shorter than with the reservoir simulator, the optimization can go on until convergence. The ANN can then be replaced by the reservoir simulator again, still using the same population that resulted from optimization on the ANN. The reservoir simulator can then be used for optimization for some generations again, generating additional and more precise training data. This cycle can be repeated until a good solution has been discovered. The point of this scheme is to take advantage of the already generated training data from optimization on the reservoir simulator in the most efficient way. Figure 3.4 illustrates how this scheme is intended to work in a general solutions space.

The following steps summarizes scheme 3.

```
1 GA-optimization is performed on reservoir simulator for
    several generations
2 ANN is trained using the data from the reservoir simulations
3 GA-optimization is performed on the ANN -> convergence
4 goto step 1 if the current best solution is not good enough
5 final round of GA-optimization is performed on the
    reservoir simulator
```

As was discussed in section 2.1.2.4 and illustrated in figure 2.3, improvements in convergence measurements happens rapidly in the start. The author

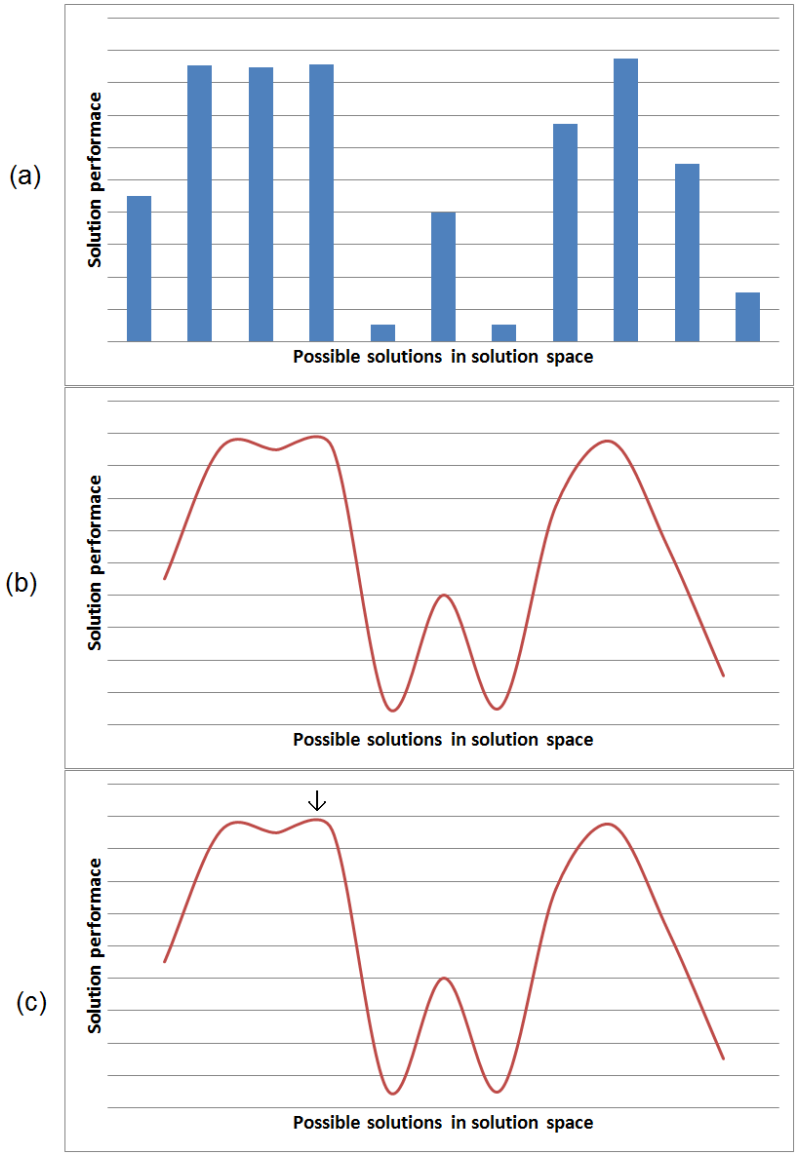

Figure 3.3: Scheme 2: (a): Training data are generated, the input parameters should be distributed along the entire solution space. (b) An ANN can be trained from the training data. (c) This ANN will perform like a continuous function which the GA can perform optimization on.

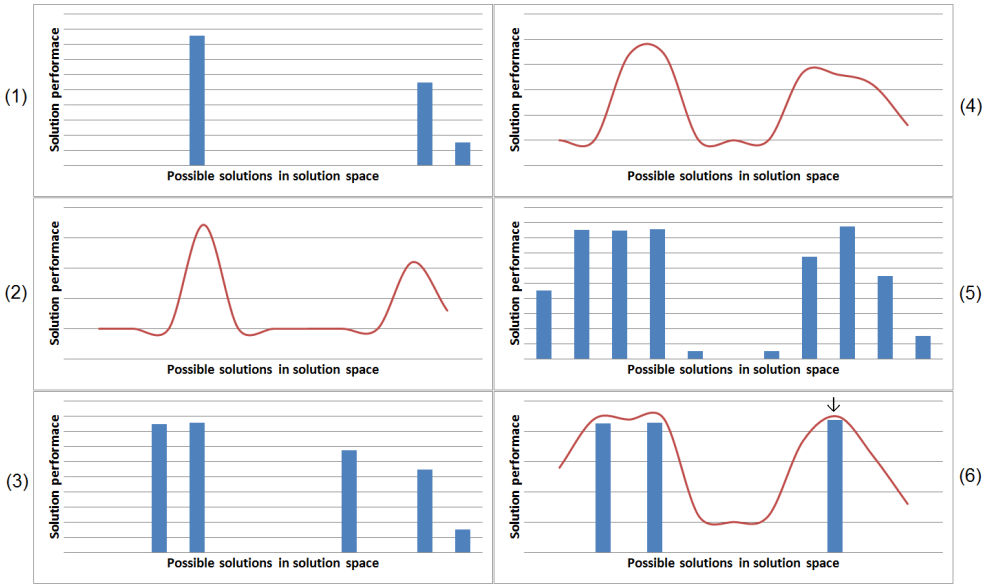

Figure 3.4: Scheme 3: (1): Training data are generated randomly by the GA (initial population) using the reservoir simulator (blue color). (2) An ANN (red color) can be trained from the initial training data. Optimization on the ANN will result in a GA population focused around the maxima in the red curve. (3) The reservoir simulator can be used to generate additional precise data around the focus areas. This additional data will be preserved along with the already existing training data. (4) A new ANN can be trained on the accumulated training data, and a new optimization can be performed on the ANN. (5) The reservoir simulator can again be used to generate precise data around the assumed optimal parameter inputs. (6) After steps 3-4 have been repeated enough times, a best solution can be located.

wants to explore the idea that in this phase a lot of good training data is generated. What follows is a period of slow development which would take a lot of time when performed on the reservoir simulator. By exploiting the training data generated, fewer simulations would have to be performed by the reservoir simulator and time can be saved.

One big challenge in this scheme is that the training data does not necessarily cover enough of the input parameter space. It will be mostly focused on a few points in the parameter input space selected as the most fit solutions by the GA. This can lead to unexpected behavior in the ANN, and potentially reduce its precision considerably. Solutions outside the explored solution space can be found to be good solutions. The algorithm should recover from this by exploring the solutions in that part of the solution space and incorporating the data in the existing training data. Figure 3.5 illustrates how the search and error-recovery can be performed on a 2-dimensional problem. The final phase of the optimization in this scheme should always be performed on the reservoir simulator rather than the ANN, to avoid faulty data resulting from the discussed potential challenges.

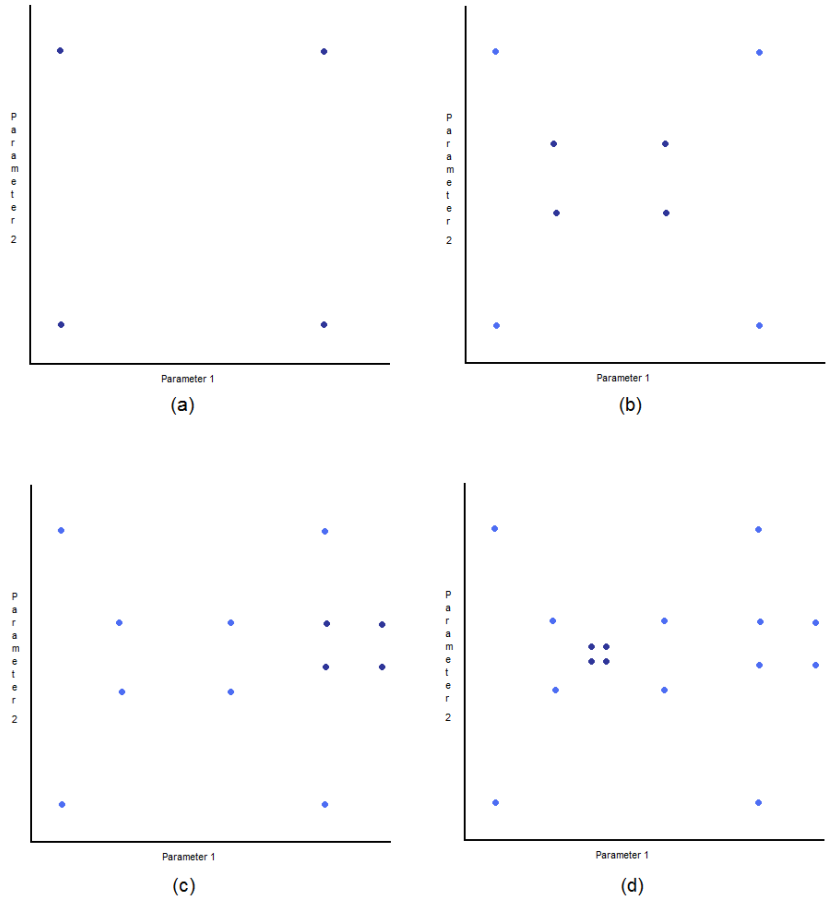

Figure 3.5: Scheme 3 in 2 dimensions: The two parameters are along the X-axis and Y-axis, good solutions are found in the ANN within the darker points for the current training data. (a): An ANN is trained from the initial training cases which covers part of the solution space. Optimization leads to improved focus in one area, the algorithm uses the ECLIPSE simulator to generate additional training data for the optimized area, and this is used for training a new ANN with good solutions within the darker points in (b). This process is repeated leading to the situation in (c) where the specialized training data has indicated a good solution can be found in a part of the solution space not previously explored. This area is then explored with the ECLIPSE simulator, and the solutions from this area are added to the training data. If this area is found to contain inferior solutions, the ANN will reflect this, and the search will get back on track like in (d).

3.2 ECLIPSE models used

This section will present and discuss the details of the ECLIPSE simulator models used for optimization. The structure and topology of the models will be illustrated and the input parameters and objectives discussed.

3.2.1 Shoebox case

This model is a simple made-up case. It has 8 input injection valves and 8 output production valves. These are aligned on each side of a 8x8 2-dimensional grid model, for a total of 64 grid cells, each with unique time-invariant geological properties. The name "shoebox" is a common description for such simple cases. Figure 3.6 is taken from the reservoir simulator and shows the details of the model.

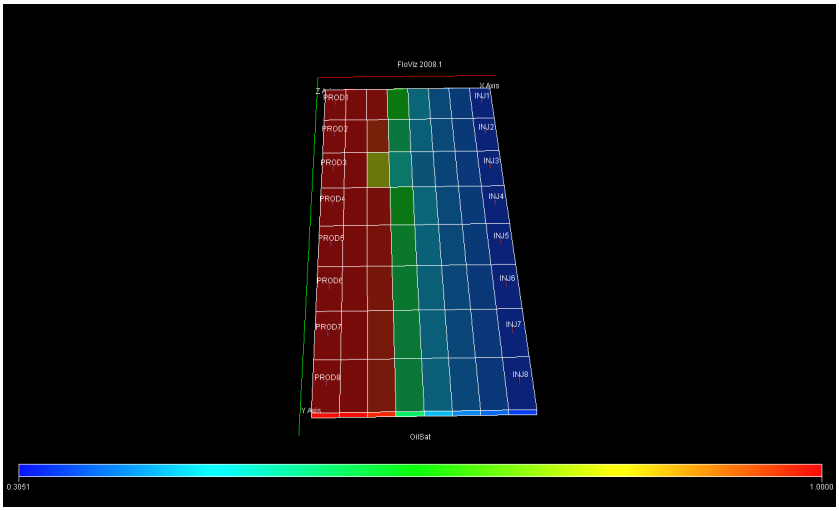

Figure 3.6: Shoebox case: This model consist of 64 grid cells. 8 injection wells are located on the right side. 8 production wells are located on the left side. The color of the tile shows the oil saturation level. Red color means mostly hydrocarbons, blue color means mostly water. This illustration is taken from the middle of a simulation run.

Several parameters can be chosen for optimization, the author has chosen the liquid flow rates for both producers and injectors as input parameters. The production horizon for this model is 6 years, and the objective for the model is the total field oil production over the whole production horizon. By adjusting the flow rates in the valves, different flow patterns are taken by the liquids and different extraction estimates can be obtained. The total oil

production is measured in standard cubic meters, Sm^3, while the flow rates are Sm^3 per day. While this model does not offer great complexity, it has served as a good source for basic information about the domain and as a testing ground for the technologies used.

3.2.2 Brugge case

The Brugge case is a realistic large-scale model that has been used for academic purposes before. There exists good solid data on optimization performance on this model already, which makes it an excellent case for the author's study. A 3-dimensional model consisting of 450000 grid cells was originally used to generate ten years of initial production history. The optimization model is an upscaled version with 60048 grid cells, and the production horizon for this model is 20 years.[11]. Figure 3.7 illustrates the topology of the reservoir model.

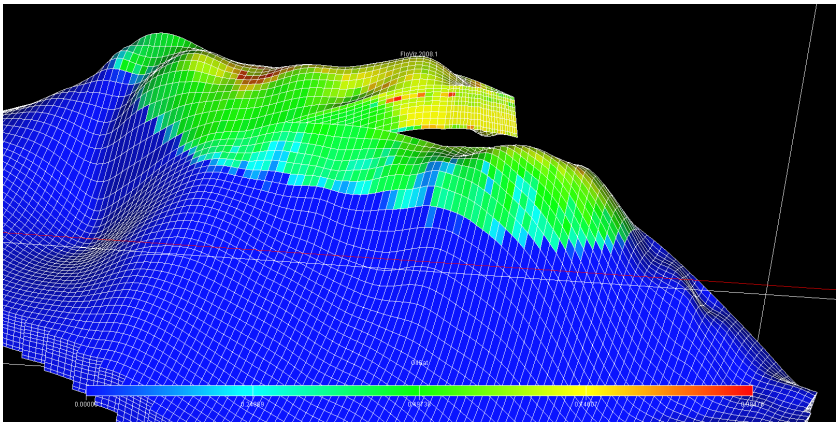

Figure 3.7: Topology for the Brugge case.

The field has 20 producer wells and 10 injector wells. Individual grid cells in the model have different geological properties and positioning as displayed by figure 3.7, making the fluid dynamics of the system complex and optimization challenging. The time for a simulation run on the Brugge case is around 85–95 seconds on the hardware available to the author. The Brugge case with wells is illustrated in figure 3.8. In this figure the height dimension has been scaled down compared to in the previous illustration.

Each of these wells are divided into 3 segments where flow can be controlled independently for each segment. This allows for 60 production rates for the producers and 30 for the injectors. Considering the values will not

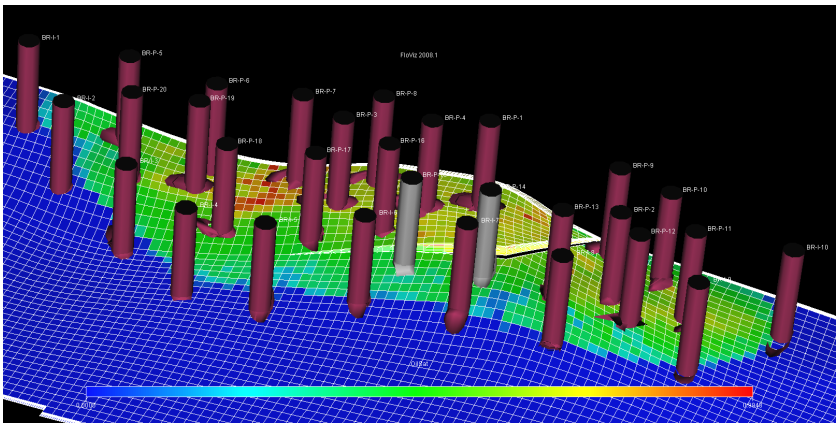

Figure 3.8: Brugge case: 10 injector wells (labels starting with "BR-I") surrounds 20 producer wells (labels starting with "BR-P") in this 3-dimensional 60048 grid cell model. Oil saturation levels are shown by the color in each grid cell.

be changed over time like discussed in section 2.2.3, optimizing for flow rates like for the shoebox case would mean 90 parameters to optimize over. A discussion of an alternative strategy for optimization follows.

3.2.2.1 Optimization strategy

Previous uses of this case have shown that injection rates can be decided by simply replacing produced volume of liquid with water [11], and this is also the scheme the author has chosen. This eliminates the injector wells from the optimization parameters. A number of segments in the producers are disabled; the total number of optimization parameters is the remaining 54 producer segments. The parameters for the remaining producers are maximum allowed water cuts[2] for each segment. At first this measurement does not make much sense, since this property of the produced liquid does not concern itself with the flow of liquids in the wells directly. The reasoning behind this choice is internal constraints in the model. These are all outside the scope of this dissertation, and the author will not discuss all of them. Lorentzen [11] provides a much more complete presentation of this case and parameters. The internal constraints are in regard to maximum well injection- and production rates, pressures in the wells and such properties. The simulator will simply produce as much liquid as is possible according

[2]Water cut is the percentage water in the produced liquid.

to these constraints. The maximum allowed water cuts in each segment will then be the limiting factor for production in the model, and will also be used to calculate profit estimations.

The optimization strategy used for this model should maximize net present value (NPV) over the next 20 years of production. Given economic parameters for NPV are 80$ income per barrel of oil produced, and 5$ cost per barrel of water produced and another 5$ cost per barrel of water used for injection in the injection wells [11].

Like mentioned earlier, each producer is divided into 3 segments with individual control for maximum water cut in each segment. Figure 3.9 shows how each segment is exposed to different levels of oil saturation/water cuts[3]. In light of this figure, the effect of considering water cuts for calculating profit

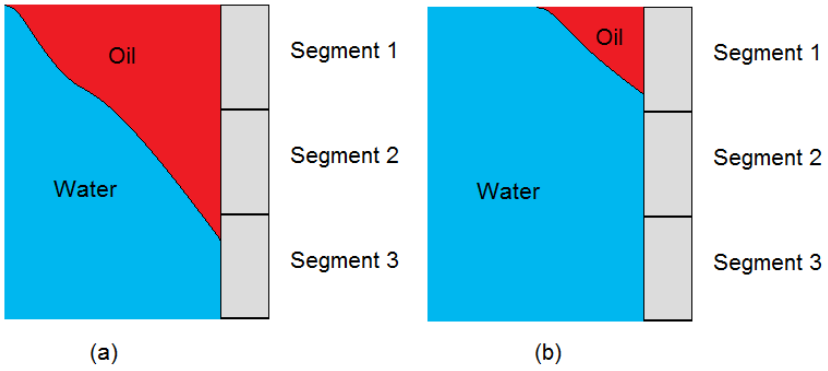

Figure 3.9: Well segments for a producer well in the Brugge case. (a) shows an oil-water front along the producer segments. While segment 1 and 2 are located well for producing only oil, segment 3 is partially flooded and will also be producing water. This water-front will move in time as oil is extracted, eventually leading to the situation in (b). In (b) segment 2 and 3 are completely flooded and producing water, while segment 1 is producing some oil and some water.

becomes more obvious. First remember that any liquid produced must be replaced by the same amount of water, at a cost. In addition, the produced water in the liquid produced (as determined by the water cut) also comes at a cost. Water cuts can be measured in real-time in each segment in the

[3]The oil saturation is the opposite of the water cut, oil saturation is the percentage of oil in the liquid before production, water cut is the percentage of water in the produced liquid.

producer wells. The different segments experience different water cuts over time during the simulation run as oil is replaced by water. By setting a maximum value for the allowed water cut in a producer segment, it will produce only as long as measured water cut is below this threshold. This means that a segment will produce at full capacity according to internal constraints until the measured water cut is above this threshold. A crucial thing to keep in mind for this strategy is the time aspect; the maximum water cut means the well segment will produce until the threshold is reached. The extraction might still be profitable after this point, but further extraction could mean less production in other segments (and other wells) that has better water cut values. There are internal constraints in the model which enforces a maximum water handling capacity in the model. Details regarding these constraints can be found in a paper by Lorentzen [11]. The author did not have to take these into account, as they were handled automatically by the simulator. The effect of these constraints can be seen as a maximum total "bandwidth'" for flow. If segments with better water cut values are given more of this available "bandwidth" the result will be higher profit than if the segments with bad water cut values are allowed to continue production. The profit of the operation depends on the total water cut in all the produced liquid, so all producer segment water cut values are dependent on each other's performance for this total calculation.

By optimizing the NPV over the water cuts as parameters, helped by the internal constraints and properties of the model, direct regard to flow rates in the well segments is avoided. Optimizing segment flow rates directly would have been much harder since there are limitations for each of these, other limitations for the wells, as well as global limitations relating to pressure in the reservoir as a whole. Considering maximum water cuts is a shortcut around all those limitations (since attempting simulation with flow rates resulting in property values outside these limitations would result in an error from the simulator). Maximum water cuts allows for the simulator itself to set all parameters regarding to flow, and serves as an on-off switch for each segment based on profit alone.

While the complexity of the choice of optimization parameters is not the main concern of this dissertation as an AI-study, the chosen parameters are all valid and much used optimization parameters for this case. The choice does not affect the complexity of the optimization methods used, as they concern themselves only with these values as meaningless numbers.

3.3 Technologies used for programming

The author was encouraged to use Matlab [14] for working with the ECLIPSE [25] model. Several Matlab scripts exist to modify ECLIPSE input files and read output files. Using these scripts is straightforward, and the author saw fit to try developing a GA and ANN in Matlab.

While both implementations did work, there were several performance issues. The time to train an ANN made in Matlab was over 1000 times longer than the same code in Java. Several issues relating to representing big numbers presented themselves as well. Numbers in the magnitude of 2^{216} needed to be represented as part of the work on the Brugge case, and standard Matlab data types does not support precision in numbers larger than about 2^{60}. For these tasks Java implementations were developed and used by the Matlab code.

The result is a mix of Matlab and Java code where all time-critical tasks and low-level representations are performed by Java code while the larger-scale organization of data is handled by Matlab.

3.4 The genetic algorithm used

General concepts for GAs were discussed in section 2.1.2.1. This section will provide a complete description of GA parameters and concepts implemented in the GA for use on the optimization cases. The following steps describes the high-level behavior of the GA and corresponds to figure 2.1 in section 2.1.2.2.

```
Genetic algorithm :
1  randomly initialize 1st generation population genotypes
2  for generation 1 to last
3     develop generation genotype population to fully
        developed phenotypes
4     evaluate all phenotypes in the population with ECLIPSE
        simulator and assign fitness to each individual
5     save generation data
6     break loop if convergence is experienced
7     select parents for recombination
8     perform crossover on winners of the selection process
9     perform mutation on winners of the selection process
10     prepare the genotype population resulting from the
        recombination for replacing the parent population
11 develop and evaluate final generation. save to generation
```

```
        data history
12 declare a winner from the recorded history data
```

The algorithm should be running for longer than strictly necessary to make sure all solution paths have been exhausted and give the algorithm time to randomly get out of local maxima. At the same time delivery dates had to be adhered to, and simulation of previously discussed cases can take considerable amounts of time, as was discussed in section 3.2.2. Realistic and predictable time frames had to be used for the optimization runs.

3.4.1 Problem representation

The internal representation of the problem is a very important factor in the performance of the GA. When applying a GA to a problem, you want it to search for solutions in a solution space [8]. This means that individuals in the population represent potential solutions to the problem at hand, and fitness can be assigned to the individuals depending on their performance solving the problem. In the simulation cases discussed in section 3.2, a vector of valve properties for oil wells was discussed as the input parameters to the models, 16 for the shoebox case and 54 for the Brugge case. When simulation is performed with these input parameters, a performance measure can be extracted from the simulation results. Parameter optimization is the easiest to attack using a GA approach[8]. The idea of gene sequences is very similar to parameter vectors. The mapping between an n-dimensional vector and individuals in the population is very simple; the individuals are the n-dimensional vectors. This is the reason for the popularity of using GAs in this application area [8].

Phenotypic representation for the shoebox case The shoebox case features 16 input parameters, 8 injector flow rates and 8 producer flow rates. These are constrained to be between 80 Sm^3 liquid per day and 120 Sm^3 liquid per day.

Phenotypic representation for the Brugge case The Brugge case features 54 input parameters. Maximum water cuts (percentages) on 54 active segments distributed amongst 20 producer wells. Being percentages, these are constrained between 0 and 1. One simplifying assumption can be made however. Section 3.2.2.1 describes the optimization strategy for the Brugge case with costs and profit estimations. A simple profit calculation shows that

there is a threshold for profit with a water cut of 0.9375[4]. Higher maximum water cut will never be profitable. This means that the water cut parameter can be constrained between 0 and 0.9375.

While the phenotypic representation of the parameters is straightforward, the genotypes that make up the low-level encoding for these can be considered slightly more complex. The genotypic representation should involve strings of bits which the genetic operators can work on, as discussed in section 2.1.2.1. The phenotypes consists of n different parameter values. Each of these n values can be represented as a binary number. To do this directly can be impractical since the number of bits needed to represent the number 120 (used by the shoebox case) is 7 ($2^7 = 128$). When this number also has to be above 80 a lot of the possible numbers represented by 7 bits are not usable by the GA. A very simple scheme makes the bit representation feasible for all such parameter constraints. The number represented by the bits is a fraction of the maximum value of the bit string, assuming a maximum length. By adding and multiplying factors to this fraction, constrained parameters can be represented. The following discussion will illustrate this.

Bit value: $0101_2 = 5_{10}$

Maximum value for a 4-bit string: $1111_2 = 15$

Fraction: $5/15 = 0.3333$

This fraction will always be a number between 0 and 1, with different step lengths depending on the number of bits used to represent the string. The parameter deciding bit string length for each input parameter is called $geneSize$ in the GA code.

Genotypic representation for the shoebox case Each parameter represented in the phenotype is represented in the genotype by a string of bits. All 16 parameters have their own bit string, and all of these are combined in the genotype, resulting in a bit string of total length $16 * geneSize$. For obtaining a number between 80 and 120 for each individual parameter the following formula is used: $80 + 40 * fraction$.

[4]Assume 80 bbl liquid produced. Water cut of 0.9375 gives 75 bbl of produced water. Produced liquid is replaced by water so the produced 75 bbl and additional 5 bbl of water are injected. NPV $= 80 * 5 - 5 * 75 - 5 * 5 = 0$

Genotypic representation for the Brugge case Each parameter represented in the phenotype is represented in the genotype by a string of bits. All 54 parameters have their own bit string, and all of these are combined in the genotype, resulting in a bit string of total length $54 * geneSize$. For obtaining a number between 0 and 0.9375 for each individual parameter the following formula is used: $0.9375 * fraction$.

The effect of this representation is that the bit string length becomes a measure of resolution for the parameters. Assume a 1-dimensional optimization case has a parameter value range between 0 and 15 and an optimal parameter value of 15. 4 bits are used to represent the parameter. If the initial value of this parameter is 0, best case for improving this parameter to its optimal value is in 4 steps (by flipping every bit in the 4-bit long string). If 3 bits are used, this number of steps is reduced to 3. However, a 3-bit string can produce 8 different possible values (2^3), while a 4-bit string can produce 16 different values. If the genetic operator works on the parameter by randomly flipping bits, the 4-bit representation will statistically take longer to reach its optimal value than the 3-bit representation, given equal starting point. This is because the step length determined by number of possible solutions decreases by a factor of 2 every time the resolution increases by 1 for the 1-dimensional problem. In an n-dimensional optimization case, change in the bit string length results in an exponential increase in possible combinations. For the Brugge case, going from a 3-bit to 4-bit representation increases number of possible solution by a factor of $1.8014 * 10^{16}$. There is a tradeoff between speed and precision that must be considered. Increased resolution and smaller step lengths in a long bit string will optimize slower than a shorter bit string with larger step length and reduced resolution. The shorter bit string will result in fewer simulations needed since the GA will not have as many possible solutions to explore. It will also mean that fewer parts of the solution space are explorable, which can result in good solutions being unreachable for the GA.

To make choices for setting these parameters, several rounds of experimentation was performed. The results of these experiments are given in section 4.5.2.

Section 2.1.2.1 described how representation plays a key role in determining how well the inheritance of traits from parent to child will be reflected in the individuals. As the above discussion has shown, the mapping from genotype to phenotype is straightforward and provides a direct correlation between genotype and phenotype.

3.4.2 Genetic operators

Two genetic operators are used on the genotypes in the GA, single-point crossover and mutation.

3.4.2.1 Crossover

The crossover operator used in the GA takes 2 parents and divides their genome bit strings in half at a random position in the string. This separation will never split a single phenotype parameter, but selects positions that preserve the structure of each individual input parameter. Half of each parent is then used to form a new individual. Crossover occurrence is decided by the crossover rate. This rate determines a percentage of the population which will serve as parents for the next generation. The genotypes of the remaining part of the population will simply be copied directly to the next generation without crossover. The crossover rate is called *crossoverRate* in the GA code.

3.4.2.2 Mutation

The mutation operator in the GA takes a genotype bit string and randomly shifts 1 bit in this string. The position of the flipped bit is random. As a result, mutation can result in changes of varying magnitude in the phenotype. If the bit string 111 is changed to 110, this results in a very small change in value (1), but if the same bit is mutated to 011 the change in value is bigger (4). Mutation will only affect one of the phenotype parameters each time it is performed; it will never affect more than one phenotype input parameter. The effect of mutation will be random, and is decided by the mutation rate. Mutation affects the whole genotype population after crossover has been performed, and individuals are randomly chosen according to the mutation rate for mutation. The mutation rate is called *mutationRate* in the GA code. Values for these rates were determined by trial in section 4.5.2, as they also are problem-specific [8].

3.4.3 Population size

The individuals in the population are the available resources the GA uses to solve the problem task. The parent population is the basis for generating new search points, therefore the population size can be viewed as a measure of the degree of parallel search supported by the GA [8]. The population size is constant through the generations. The GA involves full generational turnover, every parent is replaced by a child each generation. Population size

is problem-dependent [8] and several trials were performed in section 4.5.2 to determine population size for the Brugge case. This parameter is called $nIndividuals$ in the GA code.

3.4.4 Elitism

For small population sizes, the effect of stochastic selection can result in losing good individuals by random chance. The GA can then be set to always save a number of best individuals from one generation to the next. This parameter is called $nBestIndividuals$ in the GA code, and allows a number of individuals to be copied directly between generations.

3.4.5 Fitness

The fitness for each individual is the objective function value determined by ECLIPSE simulation for each model. For the shoebox case the fitness is the produced volume of oil in Sm^3 over the production horizon. For the Brugge case the fitness is the NPV in USD of the project over the production horizon. These are both elaborated on in section 3.2.

3.4.6 Selection mechanism

Two basic categories of selection mechanisms exists; deterministic and stochastic [8]. The author has chosen a stochastic approach, where each individual is assigned a probability of being selected based on fitness performance. Less fit individuals can also be selected to reproduce, but the chance of this happening is less than for the better fit individuals. The selection mechanism uses sigma scaling for modifying the selection pressure inherent in the fitness values by using the population fitness variance as a scaling factor. The following conversion formula is used:

$$ExpVal(i,g) = 1 + \frac{f(i) - f(\overline{g})}{2\sigma(g)}$$

where g is generation number, $f(i)$ is the fitness for individual i, $f(\overline{g})$ is the average fitness in the population for generation g, and $\sigma(g)$ is the standard deviation of population fitness. Two effects can be observed. (1) The selection pressure is lessened when few individuals have better fitness calculations than the others (high $\sigma(g)$), this results in increased degree of exploration in the initial generations of the optimization run, which prevents early convergence to a local maxima. (2) The selection pressure is increased when the population has become more homogenous (low $\sigma(g)$), small differences

in fitness will give a high probability of selection. The selection mechanism is used for parent selection.

3.4.7 Solution space database

For low mutation rates and population sizes, several individuals will be identical from one generation to the next, and even with higher values for these rates, multiple individuals with identical genotypes will be generated[5]. The initial trials showed that around 50% of the individuals generated from the evolution process were simulated more than once. By saving every simulated solution in a hash table and performing lookup check on this table before performing simulation the speed of the optimization runs were greatly improved. A lot of data was generated for the different cases through initial testing and parameter experiments, these results were preserved for later use. The ANN described in section 4.5.1 was built using such a data point database.

3.5 The artificial neural network used

General concepts for ANNs were discussed in section 2.1.3. This section will provide a description of design choices for the implemented ANN.

3.5.1 Structure

The ANN is built as a directed acyclic graph. Each node communicate over weighted links to other nodes in one direction, which is commonly referred to as a "feedforward"-ANN [15]. The nodes are organized in layers. One input layer is followed by a number of hidden layers which ultimately ends up in one output layer. Figure 3.10 shows the pattern of connectivity of an ANN developed for the shoebox case.

As was discussed in section 2.1.3.2, the structure of the ANN determines the success of the ANN by its impact on the degree of overfitting of the data. The structure of the ANNs used was determined by experimentation and discussed in section 4.5.1.

3.5.2 Node input and output

Each node takes input from all nodes in the layer before and sends output to all nodes in the next layer in the graph. For the hidden layers, the rule

[5]Non-unique genotypes can be generated a result of crossing identical parents, performing mutation twice on the same bit over 2 generations, and similar situations.

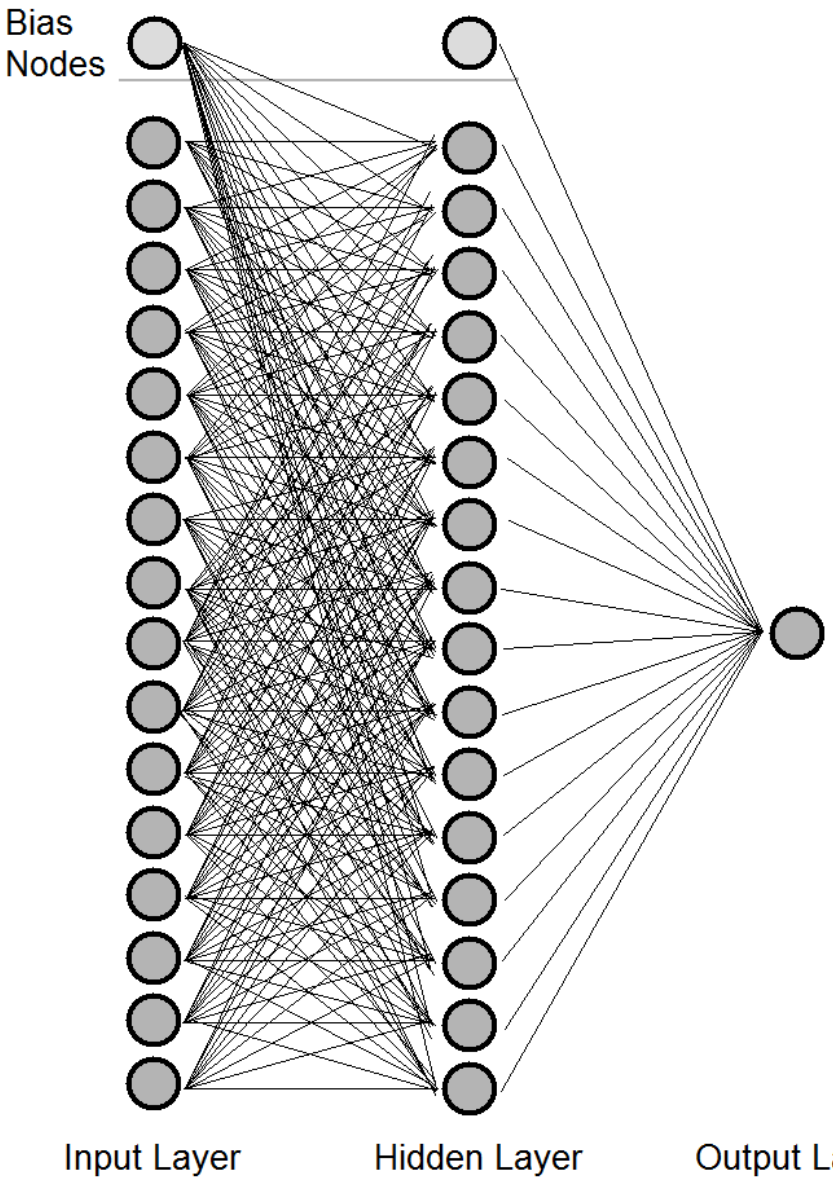

Figure 3.10: ANN developed for the shoebox case. 16 input nodes hold the input parameter values. These values are feedforward through the ANNs hidden layer and result in an output in the single node in the output layer.

for combining the inputs for node j is defined by a sigmoidal function, which produces output in a continuous range from 0 to 1 according to the formula:

$$o_j = \frac{1}{1 + exp(-net_j)} \tag{3.1}$$

The net_j total input for a node is calculated by the following formula:

$$net_j = w_0 + \sum_{i=1}^{n} x_i w_{ij} \tag{3.2}$$

For each node in the current layer, the input from all nodes x_i in the previous layer is multiplied with the weight between the nodes w_{ij}. The w_0 factor is the weight of a bias node, which has a constant activation (output) value of 1 and takes no input. For the nodes in the output layer, a linear function is used which simply sums up all inputs according to equation 3.2 and sends the result as output. For full details on these concepts, Callan [3] provides a good description.

In the trained ANNs used in this dissertation, input nodes receive their activation output directly from the solution input parameters. For both ECLIPSE models used the input parameters in the input layer of the ANNs are the same as the phenotypic representation presented in section 3.4.1. The output value of the output layer of the ANN corresponds to the objective function value produced by the ECLIPSE simulator. Input and output values in training patterns were normalized for better ANN performance. When performing simulation with new cases on the ANN after training, these values were expanded to their de-normalized form for evaluation.

3.5.3 Node error

The error of the trained network compared to the original training pattern can be calculated for each node in the ANN. For linear output nodes the error δ is calculated simply by subtracting the activation (output) value y from the target value t:

$$\delta = t - y \tag{3.3}$$

For the hidden layers this calculation is performed with the following formula for sigmoidal function nodes:

$$\delta_j = o_j(1 - o_j) \sum_k \delta_k w_{kj} \tag{3.4}$$

Where δ_j is the error for the node in question, o_j is its output, δ_k is the error of a node in the next layer, and w_{kj} is the weight between the current node j and the node in the next layer k. For elaboration on how the formula for these error estimates is derived, Callan [3] provides a complete description.

3.5.4 Weight update

When error estimates are available, these can be used to update the weighted connection between nodes according to the following formula:

$$\Delta w_{ij}(n+1) = \eta(\delta_j o_i) + \alpha \Delta w_{ij}(n) \tag{3.5}$$

Where $\Delta w_{ij}(n+1)$ is the change to a weight at step $n+1$, η is a learning rate and α is a momentum rate for learning. The momentum rate takes into consideration the weight update in step n and is there to avoid the weight updates oscillating around a value. The learning and momentum rates are parameters determining the success of the ANN. These must be tuned according to problem specifics, like number of training cases. These rates were determined by experimentation in section 4.5.1. These parameters are called *learningRate* and *momentumRate* in the ANN code.

For elaboration on how the formula is derived, Callan [3] provides a complete description.

3.5.5 Learning

The ANN employs the backpropagation algorithm for supervised learning of weights between nodes.

```
Backpropagation algorithm
1 for all training patterns
2     for each node in the input layer, assign corresponding
          training pattern input value
3     for all layers calculate input and output according to
          equations (3.1) and (3.2)
4     for all layers calculate error according to equations
          (3.3) and (3.4)
5     for all layers update weights for each node according
          to equation (3.5)
6 run training patterns on the ANN and check performance
7 if performance is not good enough, goto step 1
```

The number of times this cycle is repeated is referred to as the number of epochs of the ANN. This can also be set as a parameter for training the ANN with backpropagation, and is called *epochs* in the ANN code.

Callan [3] can provide further details on backpropagation.

3.5.6 Training data

Two methods were used for generating training data.

Random The genetic algorithm initiate a completely random population in the start, by simply generating a large enough initial population and running the GA for only the first generation, a random set of training cases were generated.

Incremental A special approach designed to spread the input data as best as possible along the solution space was also experimented with. A number of desired simulations must be specified beforehand, $nSimu$. The problem solution representation described in section 3.4.1 features parameter input vectors represented as huge strings of bits. The possible total number of values for a string of bits is $2^{bitStringLength}$. By dividing the value of the maximum value $bitMax$ (ie. 1111 for a bit string of length 4) by $nSimu$, a smaller bit string will be the result, $increment$. By starting with a bit string of value 0 $bitMin$ (ie. 0000) and then adding $increment$ to this value $nSimu$ times, $nSimu$ number of different solutions can be generated. These solutions will all be of equal distance from each other in terms of bit string numeric value. This is not the same as saying they will be of equal distance in the solution space, the success of this approach will also be seen in the experiments in chapter 4.

Chapter 4

Results and testing

In this chapter the specifics of the different optimization runs and the produced results are presented.

4.1 Proof of concept

This section presents performance results from initial optimization runs made on the shoebox case with the 3 different schemes described in section 3.1. Details on obtaining parameters for the runs are not included in this section, but are discussed for the larger Brugge case in section 4.5.

4.1.1 The GA (scheme 1)

Figure 4.1 shows the result of running optimization on the shoebox case using the GA as an optimizer on the ECLIPSE simulator. The optimization run was performed with a GA population size of 11, and ran for 120 generations. Because of the low population size, the best individual from each generation was preserved into the next generation, as was discussed in section 3.4.3.

1320 fitness evaluations were performed over the 120 generations. Of these, 1003 (75.98%) were performed simulations in the ECLIPSE simulator, while 317 were lookups of saved simulated values, as discussed in section 3.4.7. Figure 4.1(a) shows fitness for the best individual over all generations. We can see this property increasing fast in the beginning and converging to one solution at the end of the optimization run. This is the desired behavior of a GA, and shows that the algorithm works for optimization. Since the best individual is saved from one generation to the next, evolution can never produce a solution that is not better or equal to the best one in the previous generation. Figure 4.1(b) shows population fitness variance over the genera-

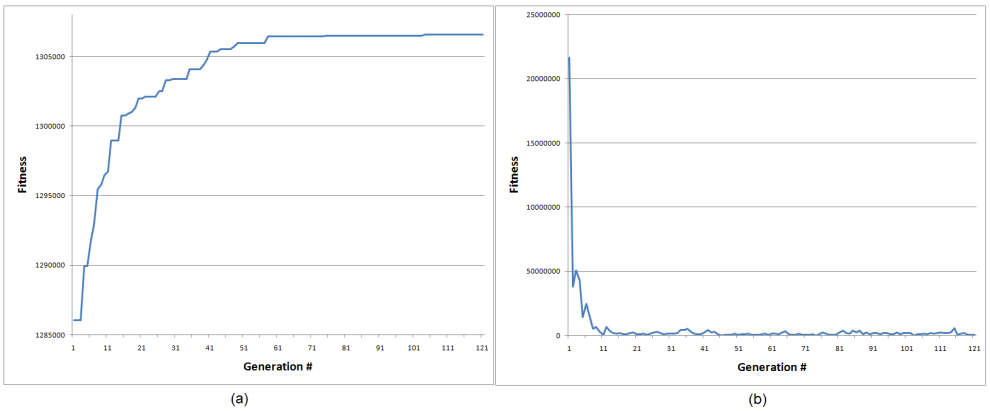

Figure 4.1: Scheme 1: (a) shows fitness for the best individual in each generation. (b) shows population fitness variance in each generation

tions. We see large fitness variance for the first generations, but after around 10 generations, this variance is low. This shows that one promising point in the solution space has been located, which is explored for the rest of the optimization run.

Optimal fitness value was found to be 1306549.4

4.1.2 The ANN (scheme 2)

For this experiment, an ANN was trained using 600 training cases generated across the solution space using the incremental approach discussed in section 3.5.6. Figure 4.2 shows the result of running optimization on the shoebox case using GA as an optimizer on the ANN. The same initial population was used, and we can see the GA performing the same way as on the ECLIPSE simulator.

Optimal fitness value was found to be 1303471.4. When running this solution on the ECLIPSE simulator, the fitness value is reported to be 1305902.1. These results are both very close to the result obtained in section 4.1.1, and shows the ANN being a good approximation of the ECLIPSE simulator.

4.1.3 ANN and ECLIPSE model (scheme3)

In this experiment, the optimization run was performed with a GA population size of 11. The 5 first generations were run on the ECLIPSE simulator.

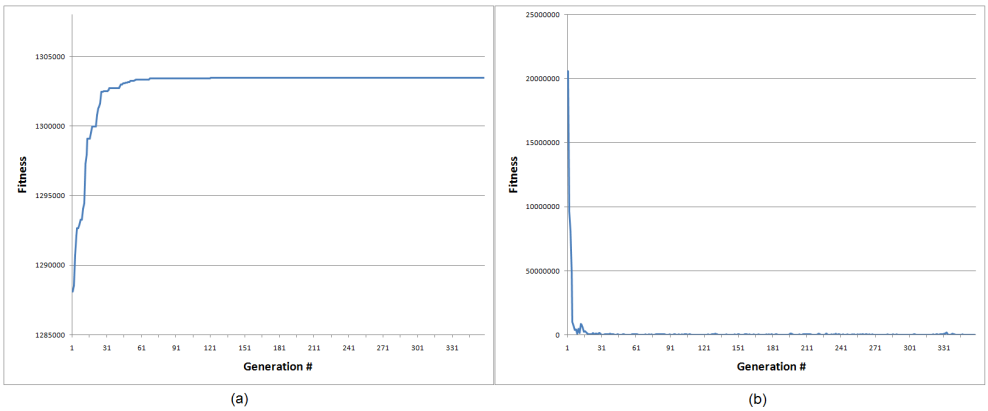

Figure 4.2: Scheme 2: (a) shows fitness for the best individual in each generation. (b) shows population fitness variance in each generation

After these 5 initial generations, the evaluated solutions were used to train an ANN, and optimization was performed for 49 generations on this ANN. After the 49 generations, the population was simulated for 1 generation using the ECLIPSE simulator. These new evaluated solutions were then added to the solution database, and this was used to train a new ANN. The new ANN was then used for another 49 generations. This cycle was repeated.

The total number of generations was 4905. Of these, 105 were performed on the ECLIPSE simulator, resulting in 955 (90.52%) simulations that had been performed on the ECLIPSE simulator and 100 lookups of saved simulations.

Figure 4.3 shows the result for all 4905 generations. The picture is a lot different than the previous fitness plots. The first 49 generations using the ANN yields very high fitness performance values. When the GA uses the ECLIPSE simulator again on the optimized results, the actual solution performance values are found to be far away from the performance values the ANN produced.

Figure 4.4 shows the same graph as figure 4.4, but only for the first 500 generations. We can see that for each 50 generation cycle the performance difference between the ANN and the actual solution performance on the ECLIPSE simulator is made smaller by the added training data until at around generation 350 the algorithm converges to one solution. The ANN produced from the explored parts of the solution space has become a good approximation of the ECLIPSE simulator for the relevant parts of the solution space.

In figure 4.5 only the generations performed on the ECLIPSE simulator is

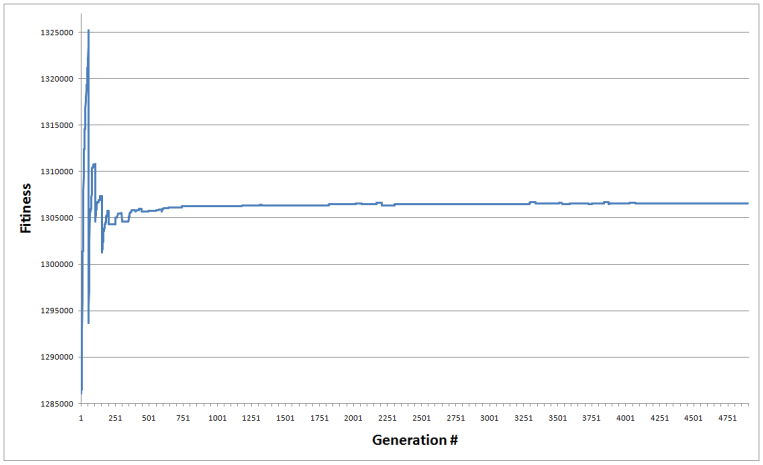

Figure 4.3: Fitness plot for scheme 3 on the shoebox case. 5 initial generations are run using the ECLIPSE simulator, these data are used to train an ANN, which the GA uses for the next 49 generation. After 49 generations using the ANN, the GA goes back to using the ECLIPSE simulator for one generation. This cycle is repeated and the algorithm converges.

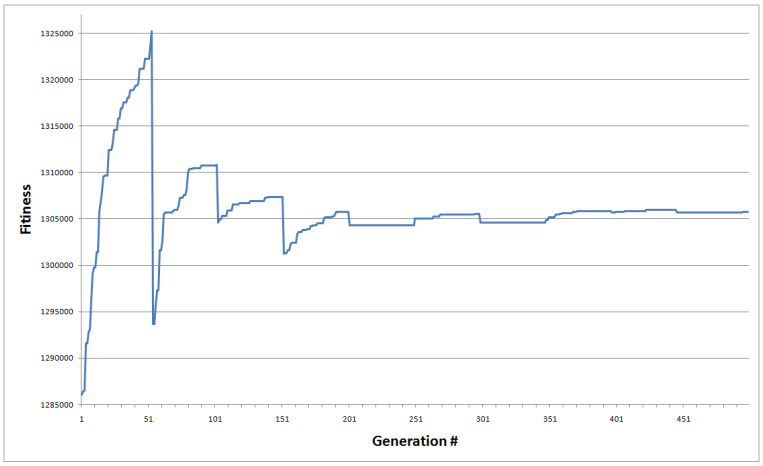

Figure 4.4: Fitness plot for the first 500 generations for scheme 3 on the shoebox case.

included, the generations run using the ANN is not included in this plot. We see fast convergence, with visible jumps in fitness measurements in the start of the optimization run. The situation at generations 7–9 shows an example of the scenario discussed in section 3.1.3 and illustrated in figure 3.5. The ANN trained from the initial data finds a solution that is far outside of the explored parameter input space. When this solution is evaluated by the ECLIPSE simulator, the performance measure does not correspond to the one performed by the ANN. When this new training data for the relevant parts of the solution space is incorporated into a new ANN, the "error" is corrected and the algorithm gets back on the right track after only one generation of repairing the inaccurate ANN approximation. The graph shows that this phenomenon occurs a few times during the optimization run, generally with a smaller effect for each occurrence than for the previous one.

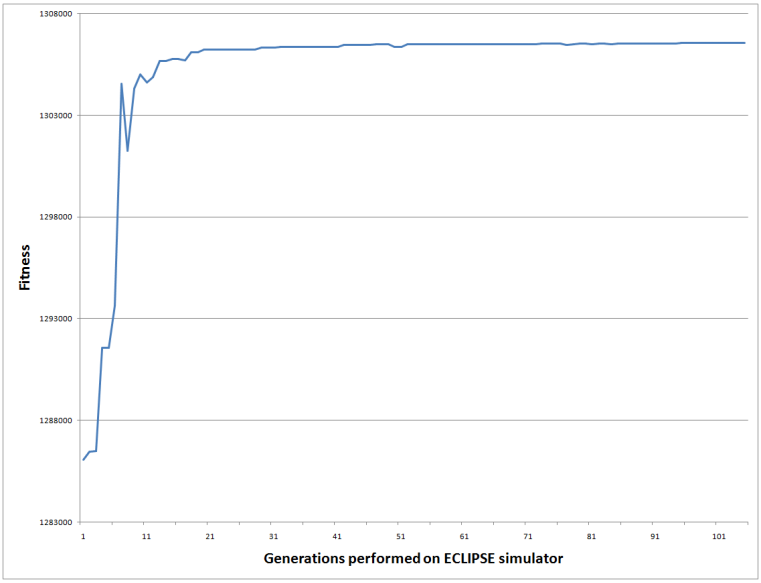

Figure 4.5: Fitness plot only for the generations run on the ECLIPSE simulator for scheme 3 on the shoebox case.

Optimal fitness value was found to be 1306549.4, the exact same result as in scheme 1.

4.1.4 Preface to optimization on the Brugge case

The experiments presented in the above sections are intended as proof of concept for the schemes presented in section 3.1. The discussion that follows

will therefore be brief compared to the more lengthy discussion in chapter 6 which concerns itself with the more thorough experiments for the different schemes on the Brugge case.

As was mentioned earlier in this section the GA appears to be performing like an optimizer is supposed to perform. Figures 4.1 and 4.2 showed convergence measurements improving until convergence is experienced in the GA optimization runs, using both the ECLIPSE simulator and the ANN.

The ANN trained for investigating scheme 2 was shown to be a successful approximation of the ECLIPSE simulator for this case. The ANNs trained in scheme 3 by using the GA training data also imbued the algorithm with an ability to adapt to the solution space in an incremental way that proved to be very successful.

We can inspect the fitness results of the different schemes together in figure 4.6. The figure shows the fitness performance for all the schemes. We

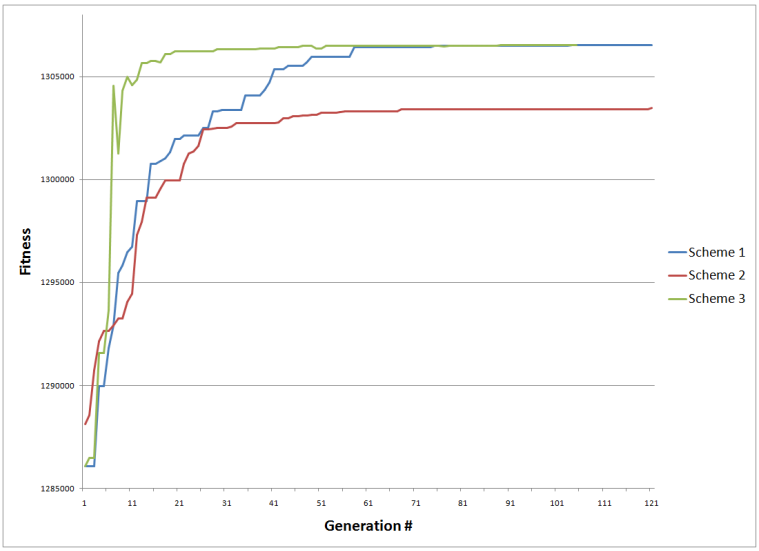

Figure 4.6: Fitness plot for all the 3 schemes on the shoebox case.

see that scheme 1 and 3 converge to the same solution. Scheme 3 converges a lot faster than both the other schemes. It is important to take into account the number of actual simulations in this comparison. Like discussed in section 3.4.7, many solution are evaluated multiple times, they are not all unique. In these cases the fitness is not found by simulation, but by lookup in a hashtable instead.

In Scheme 1, 1003 simulations were performed on the ECLIPSE simulator, and a good solution is found around generation 60, or half of the total number of simulations. It is also likely that more lookups are performed towards the end of the simulation run than in the beginning as the population becomes more homogenous. Approximately 600 simulations were performed before a good solution was established.

In scheme 2, the hashtable lookup was not used. The training data was distributed along the solution space before the optimization. 600 training cases were used, and the time needed to perform the number of generations with the GA is irrelevant when compared to the ECLIPSE simulation time. Also, it should be taken into account that the plot shows the fitness evaluation from the ANN, not the ECLIPSE simulator. When the optimal solution vector from scheme 2 was used on the ECLIPSE simulator, the fitness value was very close to the one found by scheme 1 and 3.

In scheme 3, 955 simulations were performed on the ECLIPSE simulator, and a good solution is found around generation 30. This scheme had a much higher share of simulations versus lookups compared to scheme 1 however. This is because changes introduced by minor inaccuracy in the optimization on the ANN will make the population less homogenous for every time the population is simulated by ECLIPSE. While scheme 1 got 75.98% of the fitness evaluations from the ECLIPSE simulator, scheme 3 used the simulator for 90.52% of the fitness evaluations. This difference is not enough to discredit the success of scheme 3 however. The number of generations needed for good results in the beginning of the optimization run is far superior to both the other schemes.

The number of simulations performed was similar for scheme 1 and 3, but a good solution was found earlier in scheme 3. There is a possibility that fewer training cases could have been used for the ANN in scheme 2 to produce a successful solution. It will always be hard to know how much training data will be needed beforehand, and the incremental and agile approach to generating training data in scheme 3 appears to be a working solution to that problem.

In summary, a good solution could be found around generation 30 or around 350 simulations for scheme 3 when considering the lookup factor. A very similar good solution was found around generation 60 or 600 simulations for scheme 1 when considering the lookup factor. 600 simulations were also used for generating an ANN for scheme 2. All these approaches to production optimization for the shoebox case proved successful, with scheme 3 appearing to work better than the other two.

The following sections will present the same schemes on the larger Brugge

case and will show if the discovered trends and effects are relevant for the larger case as well.

4.2 Scheme 1: GA used on ECLIPSE simulator

Scheme 1 was discussed in section 3.1.1. Choice of population size, learning rates, and similar GA parameters used were all discovered by means of trial. This process is presented in section 4.5.2.

Optimization was performed using 3 different settings for population size. Optimization runs were initiated using both random initial populations and a chosen good initial population. Good initial guesses are often available in practical reservoir optimization, and the used initial guess is very similar to what other researchers have used for the Brugge case before the author. Figure 4.7 displays the results for performing GA optimization on the Brugge case using a good initial guess and 3 different population sizes.

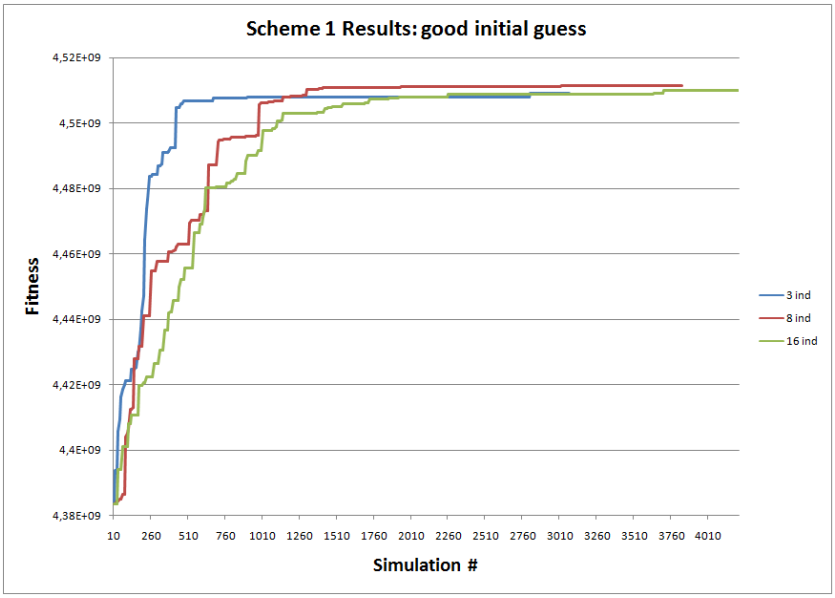

Figure 4.7: Scheme 1: Fitness plot per simulation for using 3 different population sizes when performing optimization on the Brugge case with GA. A good initial population was used.

Since the population sizes differ, the performance should not be measured per generation for comparison. The figure displays the performance of the different GA optimizations over number of ECLIPSE simulations run, where both differences in population sizes and lookup of previously simulated cases in the optimization run have been taken into account. Very low population

sizes were used, this is due to the findings in section 4.5.2.2 and discussion in chapter 6.

We can see that the GA using population size of 3 individuals is converging to a good solution before the optimization runs using 8 and 16 individuals. This solution is not as good as the ones eventually developed by the runs with larger population sizes. The convergence to a good solution for the GA using 8 individuals is also slightly faster than the GA using 16 individuals.

Figure 4.8 shows the performance of the same optimization runs using random initial populations. The same random seed was used when initializing the populations, and all GAs started with the same best solution fitness in the initial generation.

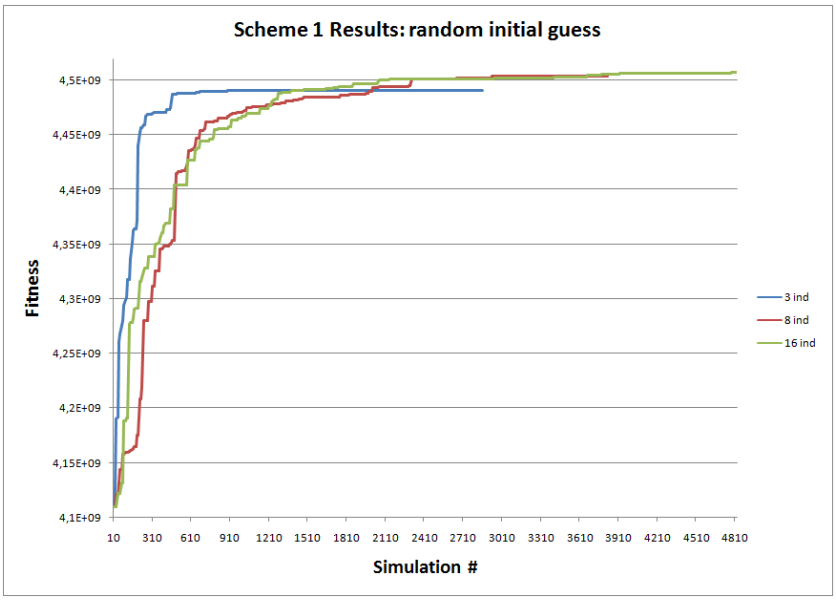

Figure 4.8: Scheme 1: Fitness plot per simulation for using 3 different population sizes when performing optimization on the Brugge case with GA. A random initial population was used.

The figure shows very similar results to the experiments using better initial populations. The GA using 3 individuals per generation converges very rapidly, but appears to get stuck in a local maximum. The other solutions both converge slower, but also more steadily and eventually discovers better solutions than the GA using lower population size. The relative difference between the GAs using 8 and 16 individuals per generation is smaller for the runs using random initial populations than for the ones using good initial

Simulation no.	3 individuals	8 individuals	16 individuals
1	4383588513	4383588513	4383588513
100	4421070884	4405027379	4401200794
240	4476540113	4440980315	4422360465
460	4505736558	4462850835	4452159401
720	4507716014	4494644696	4494644696
1000	4507777715	4506297306	4491430230
1300	4507840765	4508490101	4502895952
2120	4507840996	4511079060	4507803813
3020	4509211987	4511276768	4508870522
3710		4511276768	4509844495

Table 4.1: Scheme 1: Fitness performance for different population sizes running GA optimization on the Brugge case.

populations.

We can inspect the numerical values of the solutions for different simulation numbers in table 4.1. These results are for the GA optimization using a good initial guess. The solution fitness discovered after 460 simulations by the GA optimization using 3 individuals is not matched by the one using 8 individuals until after 1000 simulations have been performed, and even later for the GA optimization using 16 individuals. The numerical difference between the fitness of the solution discovered after 460 simulations using 3 individuals and the fitness of the best solution discovered after 3020 simulations using 8 individuals is 5540210. The result found after 460 simulations is only 0.123% worse than the result found after 3020 simulations. However, if we go back to the meaning of this number discussed in section 3.2.2.1, this value is actually the NPV in US Dollars. That is 5.5 million USD for the time and resources it takes to do additional 2600 simulations.

The solution vectors discovered by the 3 different GA optimizations can be inspected in figure 4.9. In this figure, each column represents a optimization parameter in the solution vector, and the height represents the value of the parameter.

The meaning and implications of these findings are all discussed further in chapter 6.

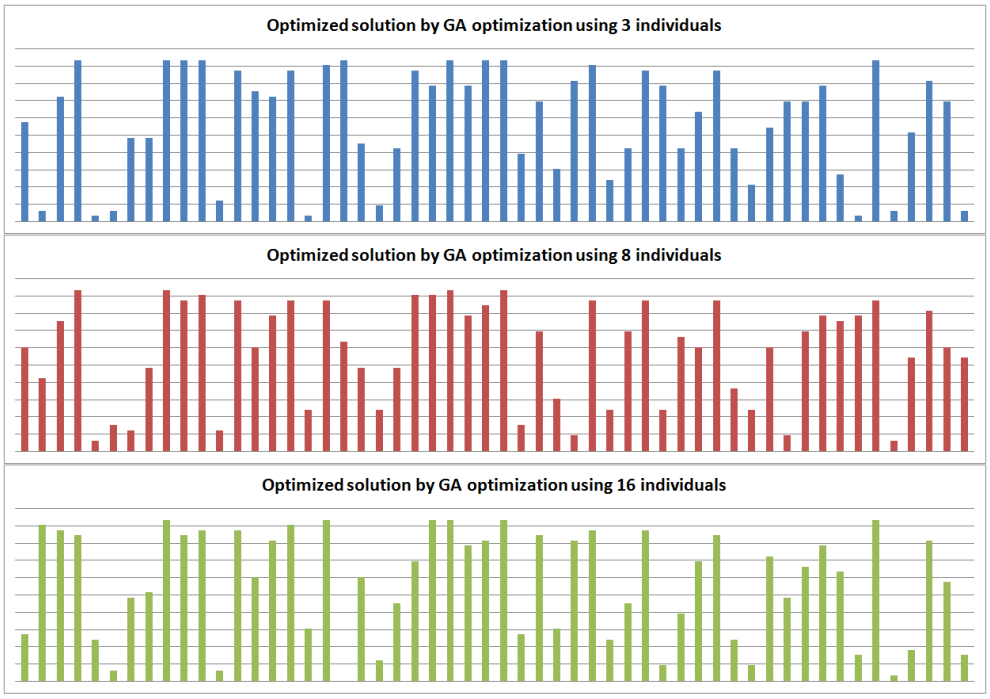

Figure 4.9: Scheme 1: Solution vectors for the optimized solutions discovered by performing GA optimization with different population sizes. Columns represent optimization parameters, and the column height represents the value of the parameter.

4.3 Scheme 2: GA used on ANN

Scheme 2 was discussed in section 3.1.2. Choice of GA and ANN parameters used were all discovered by means of trial. This process is presented in sections 4.5.2 and 4.5.1. The number of training cases used was inspired from the performance of optimization using scheme 1. To be able to compare the performance of the schemes, similar number of simulations were used.

GA optimization was performed on different ANNs trained with 200, 400, 800 and 3000 randomly generated training cases, and again for the same amount of incrementally generated training cases. These approaches were presented in section 3.5.6. Figure 4.10 shows the performance for GA optimization on ANNs trained with the different number of training cases. All optimization runs were initiated and run with the same random seed.

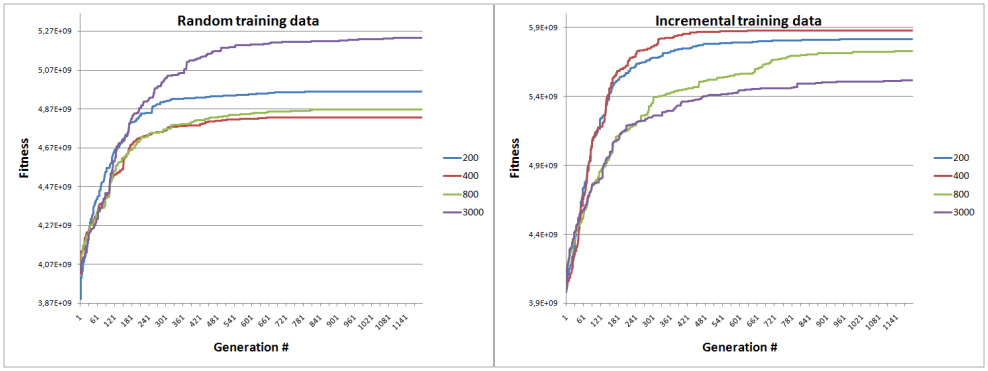

Figure 4.10: Scheme 2: Fitness performance for running GA optimization on ANNs trained with different number of training cases from the ECLIPSE simulator.

The point of scheme 2 is to evaluate the solutions with ANNs instead of the ECLIPSE simulator. The solution vectors for the initial populations in the different runs is identical due to the random seed, and the figure shows that the value of these initial guesses are similar when evaluated by the trained ANNs. We can also see all the optimization runs converging to solutions with higher fitness values than the actual values discovered by scheme 1 in the previous section. When we apply the final optimized solution vectors from the optimization runs to the ECLIPSE simulator we get the real values for the solution vectors. These are shown together with their ANN counterparts in figure 4.11.

The values evaluated by the ECLIPSE simulator is shown by the red

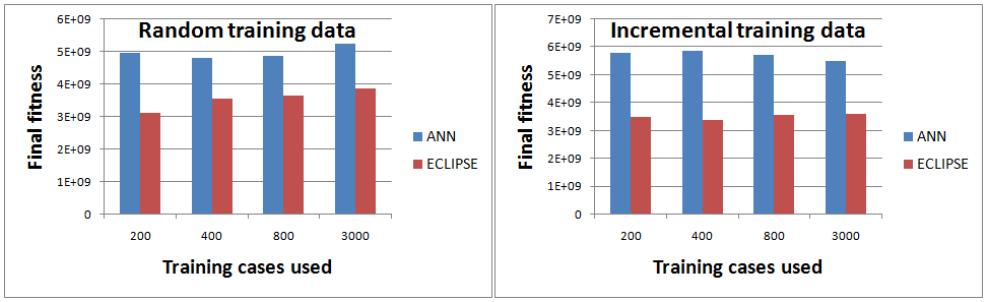

Figure 4.11: Scheme 2: Optimized fitness values from the ANNs and the real values of these solutions when evaluated by the ECLIPSE simulator.

#	ANN init.	ECL. init.	%	ANN opt.	ECL. opt.	%
Random training data						
200	3889599384	4020186350	-3.25	4959519735	3129661975	58.47
400	4022223539	4020186350	0.51	4826720299	3560127420	35.58
800	4082148403	4020186350	1.54	4867656516	3665899804	32.78
3000	4044972140	4020186350	0.62	5234528538	3872832355	35.16
Incremental training data						
200	4019740294	4020186350	-0.01	5814950499	3498988529	66.19
400	4004347450	4020186350	-0.39	5876672093	3395715210	73.06
800	4027680797	4020186350	0.19	5724885790	3567634826	60.47
3000	3982082850	4020186350	-0.95	5513610728	3592944642	53.46

Table 4.2: Scheme 2: Initial and optimized fitness values from the ANNs and the real values of these solutions when evaluated by the ECLIPSE simulator.

bars in the figure, the blue bars are the fitness evaluations performed by the ANN. For all numbers of training cases and types of training data used, the fitness evaluations of solutions by the ANNs are far away from their actual values when evaluated by the ECLIPSE simulator. The difference between the ANN and ECLIPSE evaluations is especially large for the incremental training data used. Table 4.2 shows the numerical fitness values for initial guess and optimized solutions evaluated by ANNs and ECLIPSE. The table also shows the difference between the fitness values of the solutions when evaluated by the ANN and by the ECLIPSE simulator in percentages.

For the incrementally generated training data the optimized values are over 50% higher than the actual values from the ECLIPSE simulator. The randomly generated training data performs slightly better, but still with optimized fitness far off the actual values. ANNs trained with higher number

of training cases generally produced solutions with better performance when evaluated by the ECLIPSE simulator. It should be noted that all the variations produced optimized solutions with fitness values worse than the initial guess, when evaluated by ECLIPSE.

Another very interesting point to note is the low difference between the ANN and ECLIPSE values for the initial guess. As the table shows, this difference is significantly lower than the results from the optimized solution for all variations of the optimization. The initial population solutions were not part of the training data used to develop the ANNs.

The meaning and implications of these findings are all discussed further in chapter 6.

4.4 Scheme 3: GA used on a combination of ECLIPSE simulator and ANN

Scheme 3 was discussed in section 3.1.3. The GA and ANN parameters used were all discovered by means of trial. This process is presented in sections 4.5.2 and 4.5.1.

Scheme 3 optimization was performed using a good initial population as well as a random initial population. GA optimization was run on the ECLIPSE simulator for 25 generations with a population size of 3. Using the available results, an ANN was trained, and GA optimization was performed on this ANN for 100 generations. The GA then switched back to using the ECLIPSE simulator for solution evaluation for another 25 generations, generating additional training data for training ANNs. This cycle was repeated until 4925 generations totally had been performed.

Figure 4.12 shows the results for performing scheme 3 optimization on the Brugge case using both good and random initial guesses. The figure shows the

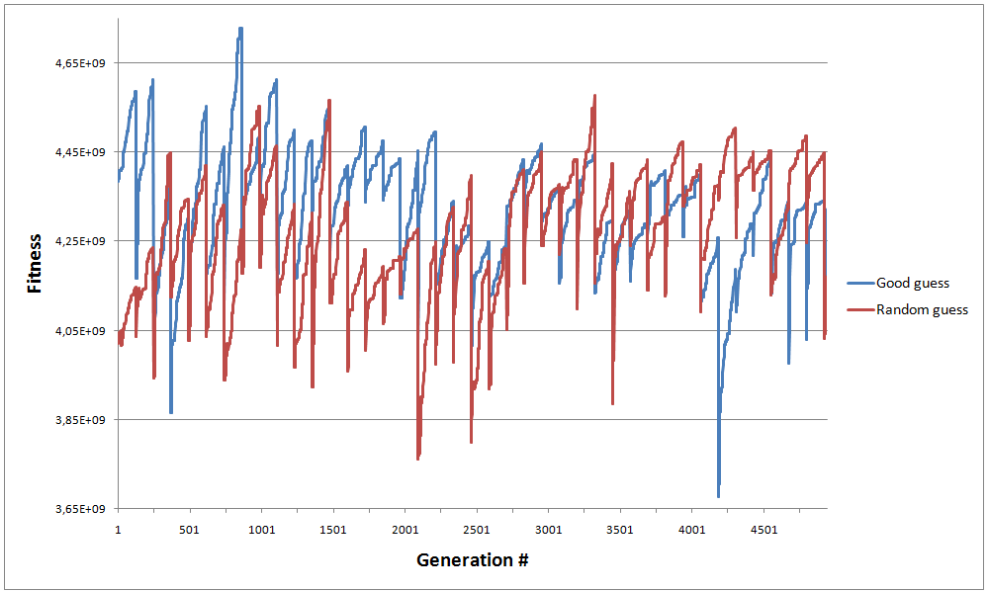

Figure 4.12: Scheme 3: Performances for scheme 3 optimization on the Brugge case using a good initial population and a random initial population.

GA improving solutions for the first 25 generations. When the trained ANNs are used to evaluate the solutions, improved solutions are found that, when

evaluated by the ECLIPSE simulator, does not have fitness values close to the ANN approximations. When this cycle is repeated the same happens over again. Optimization using ECLIPSE improves the solutions, but the ANN approximation identifies solutions that are outside the explored parameter space and evaluates these with fitness values that are not corresponding to the real model. It is not clear from the graph, but it is interesting to note that when optimization switches from ECLIPSE evaluation to ANN evaluation, the ANN approximation are very accurate for the solutions that have already been evaluated by the ECLIPSE simulator. It is in the unknown parts of the solution space the ANN approximations are inaccurate.

Figure 4.13 shows the generations performed on the ECLIPSE simulator only. No consistent improvement to the solutions can be seen from looking

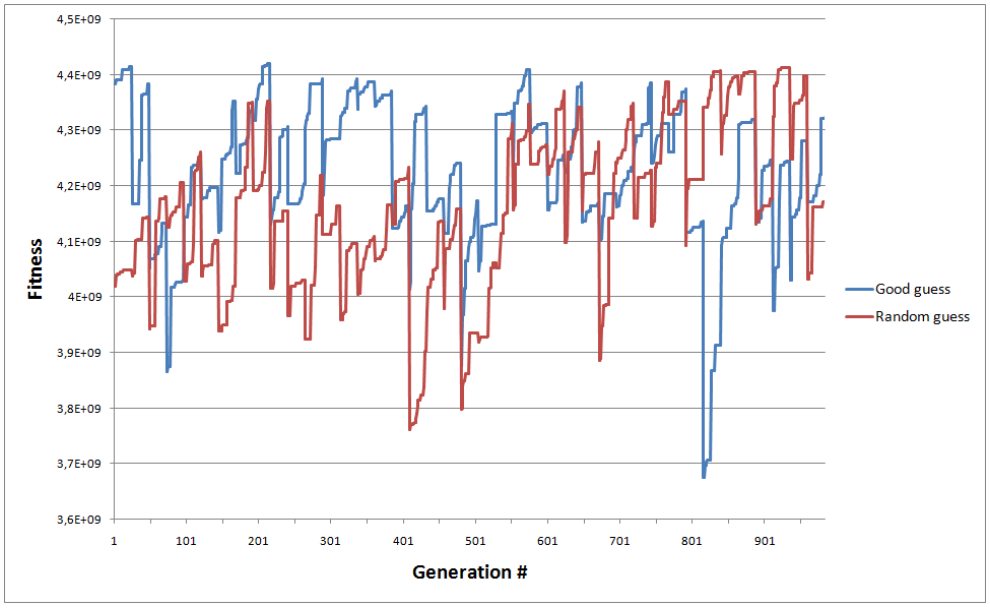

Figure 4.13: Scheme 3: Performances for scheme 3 optimization on the Brugge case using a good initial population and a random initial population. Only generations performed on the ECLIPSE simulator is shown.

at this figure. For the good initial population the optimization results in an optimized result that is worse than the initial guess. For the random initial guess the final result is slightly better. However, as can be observed in the figure, both optimization runs explored solutions that were both significantly worse and better than either initial or optimized solutions. The discovery of both good and bad solutions appears seemingly at random, and no exceptionally good solutions were ever happened upon.

The author did perform several other experiments with scheme 3, changing number of generations for using both ECLIPSE and ANNs, GA parameters and other factors. The results were very similar for all variations and these results have been omitted for the sake of brevity.

The meaning and implications of these findings are all discussed further in chapter 6.

4.5 Determining optimization run parameters

When running and preparing the GA and ANN, a number of parameters can be set that will control the performance of the algorithm, the speed of convergence, accuracy and other properties of the methods. These parameters can all be tuned to yield different results, dependent on the case, available training data, complexity of the problem representation, and other properties of the problem in question. Finding the optimal parameters is difficult, and several textbooks will tell that it is highly problem-dependant [3] [8]. Finding optimal algorithm parameters is important because the performance of the algorithms is highly dependent on the parameters chosen.

Optimization is the topic of this dissertation. It concerns itself with optimization of production in oil reservoirs. The author has chosen to tune GA and ANN parameters independently. This is risky since they are all dependent on each other. A dissertation could be concerned only about finding optimal parameters for ANNs and GAs, and most textbooks do cover this. Consequently, the author has avoided putting too much focus on this optimization. When optimizing the algorithm parameters, a starting point was chosen according to the recommendations of mentioned textbooks and the authors own experience. Several values for the parameters were experimented with, and the results were thereafter inspected. In this way, the optimization of the algorithm parameters was performed manually by trial and error. This process is documented in the following subsections.

Descriptions follows of how the GA and ANN parameters were determined for use in the 3 production optimization schemes described in the previous sections. The ANN will first be discussed as it was later used to assist in determining the parameters for the GA.

4.5.1 ANN parameters

As was discussed in section 3.5, there are several different parameters that determines the performance of training in the backpropagation algorithm, and consequently, the success of the ANN as an approximation of the ECLIPSE simulator. As is stated by Callan [3], "The application of neural networks is an experimental approach to engineering.". The author has taken available guidelines into consideration, and established all algorithm parameters for training the ANNs by trial and error.

In the following experiments, 2 sets of training data were used. The number of training cases to use was inspired by the results found for scheme 1 in section 4.2 and discussed in section 6. The number of training cases was set to 800 since good solutions could be produced at that number of

simulations in scheme 1. The ECLIPSE simulator was used to produce 800 training cases randomly and 800 cases in the incremental way. Both these approaches were discussed in section 3.5.6. Experiments were run with both sets of training data, and the solutions were then evaluated by running the other data set as testing data for the ANN produced.

4.5.1.1 Learning rate

For the first experiments, different learning rates were experimented with. The learning rates explored were 0.1, 0.125, 0.13, 0.14, 0.15 and 0.2. The structure of the ANNs used consisted of 54 input nodes, 54 hidden nodes, and 1 output node, corresponding to the input–output sets of the ECLIPSE simulator. Training was run for 500 epochs using a momentum rate of 0. All these parameters are explored and tuned in the following subsections. Figure 4.14 displays the results for training with the different learning rates.

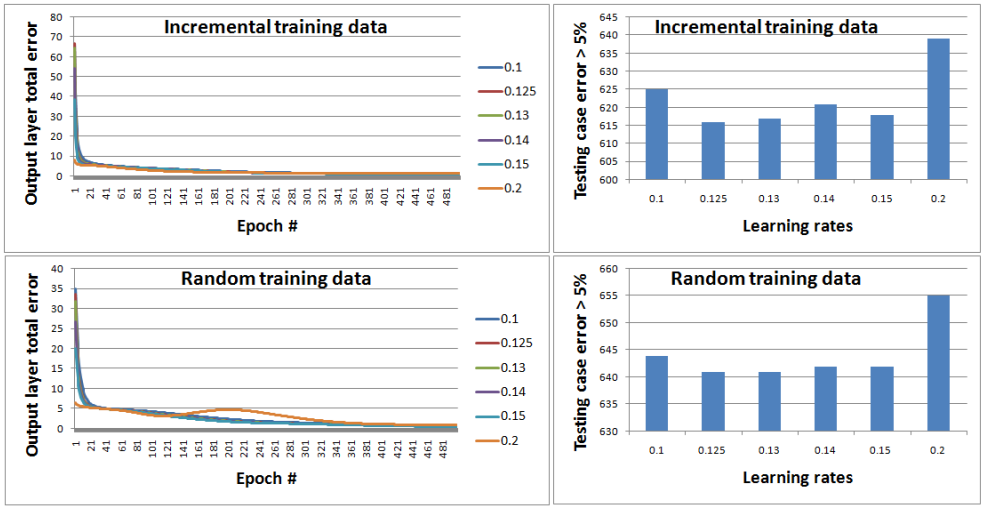

Figure 4.14: Performances for training ANNs with different learning rates. The left-hand graphs show the total error for all training patterns in the output layer for each epoch. The right-hand charts show number of incorrect cases after running the testing data set on the ANN, using 5% error tolerance.

The figure shows a graph of total error for all training cases in the output layer for each epoch and a chart of number of incorrect cases after running the testing data set on the ANN with 5% error tolerance. This is displayed for both sets of training data. The chart shows that similar results were obtained with all learning rates, with slightly better performance at learning

rates around 0.13. The graphs show total error in the output layer converging to very low values for all learning rates. Figure 4.15 shows the accurate development for the 20 last epochs.

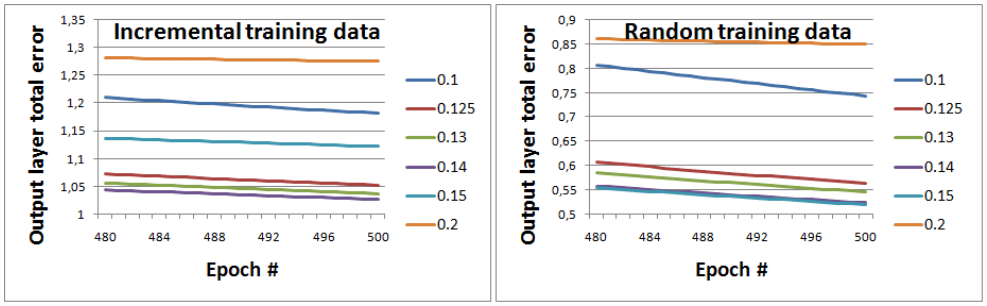

Figure 4.15: Total output layer error for training ANNs with different learning rates for the last 20 epochs.

The figure shows total error is low for learning rates of 0.13 and 0.14. Based on the results in the graphs and the charts both, 0.13 was chosen as the default learning rate for training ANNs.

4.5.1.2 Structure of the ANN

The results of the experiments to determine appropriate ANN structure can be inspected in figure 4.16. Training was run on each structure for 500 epochs with learning rate 0.13 and momentum rate 0.0.

For the incremental training data set, a single layer of 114 hidden nodes performed best. Using the random training data set, the best performance was found using 2 hidden layers, the first containing 100 nodes, the second containing 50 nodes. The results obtained with this structure for random training data performed better than all other tested structures.

Total output layer error for single and double hidden layer per epoch is shown in figure 4.17. Only the best results for single and double hidden layer are displayed. We see the error decreasing faster and to a lower level in the experiment using a single hidden layer. Figure 4.16 showed that this structure was inferior to the double hidden layer structure when performing on the testing data however. The ANN trained using a single hidden layer is a case of overtraining caused by too few hidden nodes. The error is low for the training cases, but the structure is not complex enough for prediction on the testing data, this is discussed further in chapter 6.

Two hidden layers using 100 and 50 nodes was chosen as the default structure for training ANNs.

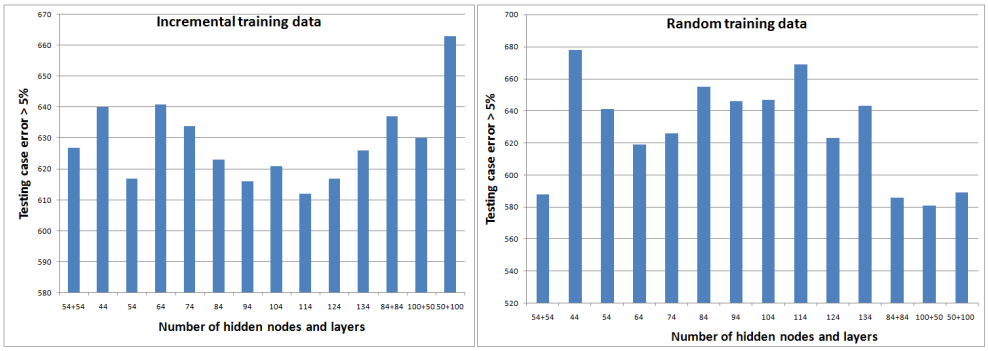

Figure 4.16: Performances for training ANNs with different structures. Single numbers on the X-axis means one hidden layer with the displayed number of nodes in it. 100+50 means 2 hidden layers, the first using 100 hidden nodes, the second using 50 hidden nodes. The charts shows number of incorrect cases from the training data set after running the data on the developed ANN, using 5% error tolerance.

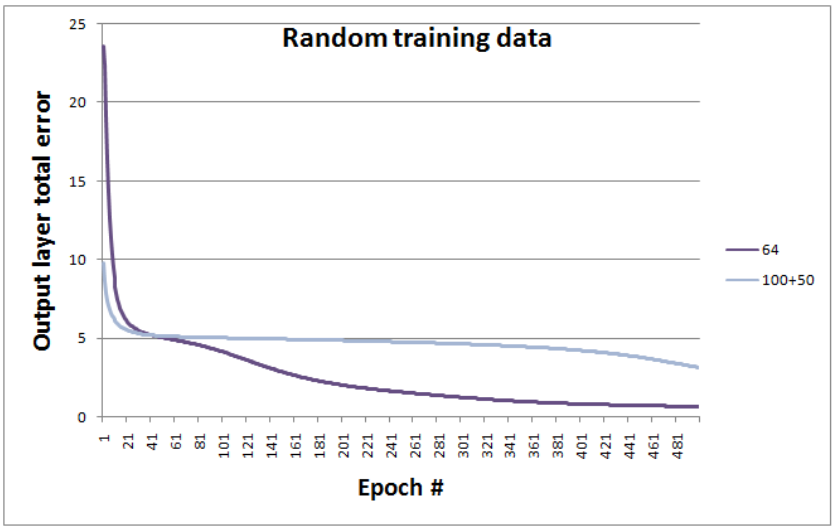

Figure 4.17: Performances for training ANNs with different structures. The graph shows total output layer error on the training cases for each epoch.

4.5.1.3 Momentum rate

Figure 4.18 shows the results of performing training of ANNs using momentum rates of 0.0, 0.01 and 0.1. The experiments used learning rate 0.13, structure 100+50 and ran for 500 epochs. The results are quite clear. A

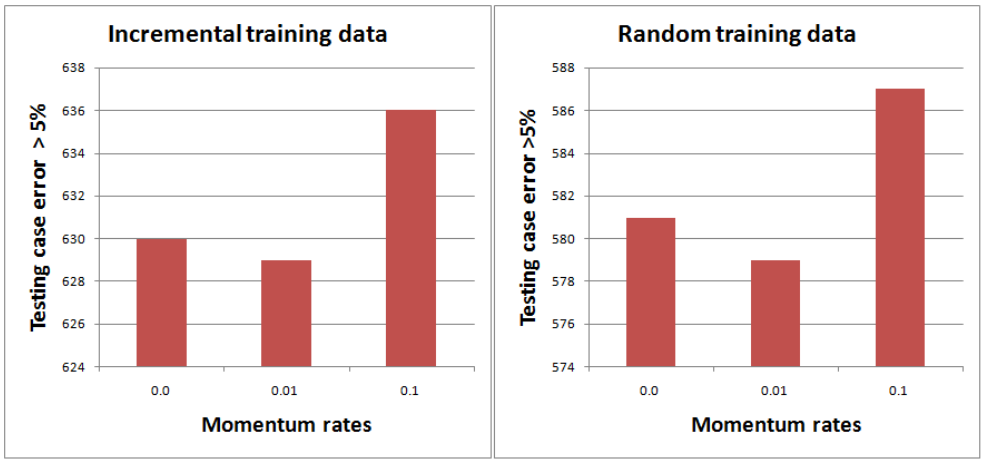

Figure 4.18: Performances for training ANNs with different momentum rates. The charts shows number of incorrect cases from the training data set after running the data on the developed ANN, using 5% error tolerance.

momentum rate of 0.01 performed slightly better than 0.0, and far better than 0.1. Momentum rate of 0.01 was chosen as the default momentum rate for training ANNs. The error in the output layer for each epoch was very similar for all cases.

4.5.1.4 Number of epochs

Figure 4.19 shows the results of performing training of ANNs using different number of epochs. The default parameters established in previous sections were all used for these experiments. The results were very similar for using both sets of training data, the results using the random training data set is shown. The chart shows that the initial guess of 500 epochs proved to be appropriate and was chosen as the default number of epochs for training ANNs.

4.5.1.5 Number of training cases

The point of the final round of ANN experiments is to determine the effect of number of training cases for the performance of the ANN, using the es-

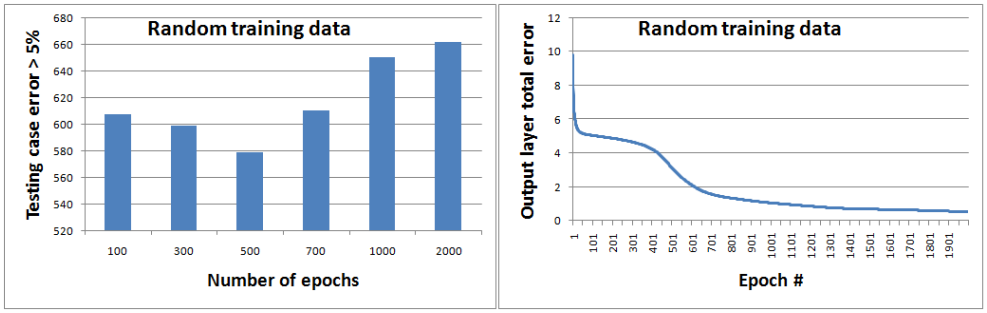

Figure 4.19: Performances for training ANNs with different number of epochs. The chart shows number of incorrect cases from the training data set after running the data on the developed ANN, using 5% error tolerance. The graph shows total output layer error on the training cases for each epoch.

tablished ANN parameters. Training was run using 20, 50, 100, 400, 800 and 4000 training cases, all randomly generated. Figure 4.20 shows the test data precision of trained ANN with the different number of training cases.

Not unsurprisingly, using very few training cases leads to poor precision. It is interesting to note that using more than 400 training cases does not improve the performance of the ANN very much. This and other aspects of the established parameters will be discussed further in chapter 6.

4.5.1.6 Training data evaluation

This subsection provides information about the training cases used for training ANNs. The two approaches for generating training data were discussed in section 3.5.6. 4000 training cases were generated for each approach used, and parts of these training data sets were used when training ANNs in the previous sections. Figure 4.21 shows the fitness for all generated solutions for both approaches in descending order. We can see that the majority of training cases have fitness values ranging between 3.65E+9 and 4.05E+9. Figure 4.22 shows the distribution of training data solutions below, in, and above this range of fitness values.

85.05% of the training cases for the randomly generated training data falls into this value range. For the incrementally generated training data this value is 84.58%. Only 7.23% of the training cases was evaluated to fitness values above 4.05E+9 for the randomly generated training data. 6.88% of the training data set had fitness values evaluated above 4.05E+9 for the incrementally generated training data.

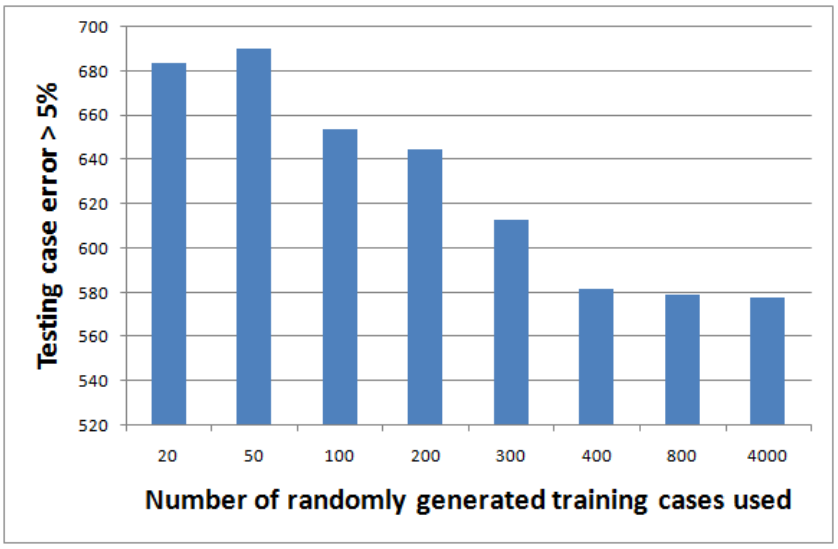

Figure 4.20: Performances for training ANNs with different number of training cases. The chart shows number of incorrect cases from the training data set after running the data on the developed ANN, using 5% error tolerance.

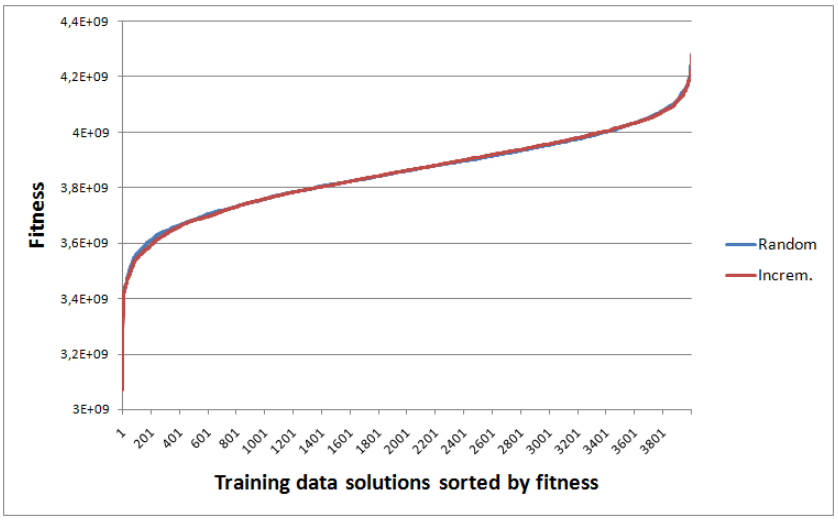

Figure 4.21: Fitness for all generated training data solutions. The solutions are sorted in descending order.

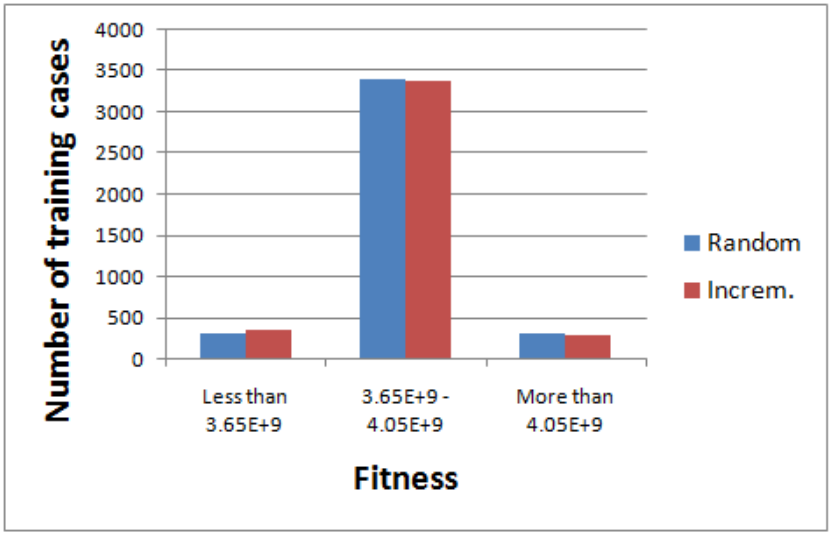

Figure 4.22: Distribution of training data over fitness values.

4.5.2 GA parameters

While use of GA for optimization problems is often problem-independent [8], section 3.4 discussed how parameters for the GA were not. This section presents and discusses the experiments performed to find good GA parameters for optimization of the Brugge case model.

Performing these experiments with the ECLIPSE simulator would require a lot of time because of the high simulation time using the simulator. The author elected to not use the ECLIPSE simulator for the experiments, but rather use a different technique much used in this dissertation. An ANN was trained using previously collected data points from the ECLIPSE simulator. This ANN was then used to perform the experiments to establish good GA parameters.

27212 data points that had been collected as part of the initial testing of the algorithms on the reservoir simulator produced an ANN with behavior similar to what can be expected of the reservoir simulator itself, even if it might not be able to provide accuracy in all parts of the solution space. Included in this training data set was cases assumed to be scattered across as much of the solution space as possible. 8000 of the total training cases had been obtained by using the methods discussed in section 3.5.6.

The results obtained from the experiments are presented in the following subsections and discussed in chapter 6.

4.5.2.1 Genetic representation

As was discussed in section 3.4.1, the number of bits used in the genetic representation of the solutions can have an impact on the precision and performance of the GA. Few bits means there are less possible solutions to explore. It also means potential good solutions are not reachable to the algorithm. Experiments were performed with different bit string lengths to determine the effects of the different choices. The results of the experiments are shown in figure 4.23. All the optimization runs were performed with a population size of 60, crossover rate of 0.3, and mutation rate of 0.5. These were then tuned in later experiments.

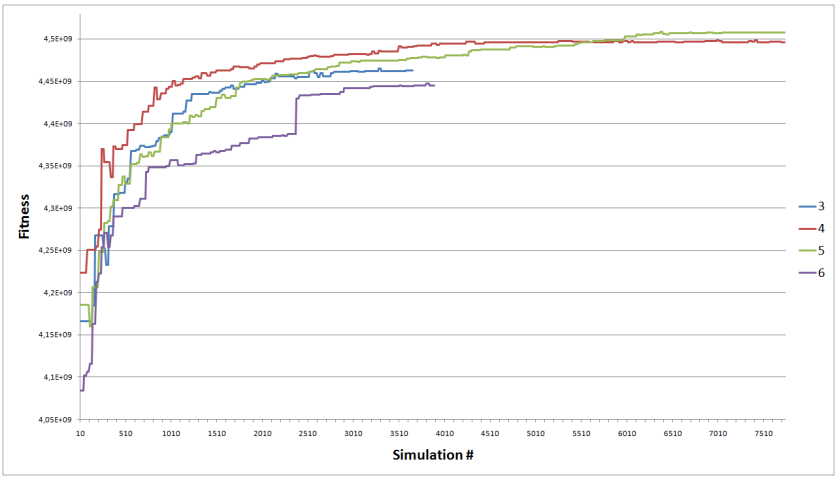

Figure 4.23: Performances for performing optimization with the GA using different bit string lengths for the genetic representation.

These optimization runs were performed on the actual ECLIPSE model, and not the ANN discussed in the introduction to this section. The figure shows the results for bit string lengths 3–5 to be similar, and somewhat worse performance from the run with bit string length 6. While lower bit string length seems to be performing better than higher bit string lengths, we see bit string length 5 converging on a better solution than the other runs. The result that was reached has the same NPV objective value as the results presented by Lorentzen [11] which are assumed to be near optimal. Since the run with bit string length 5 was the only one to reach this solution with comparable performance to lower bit string lengths, 5 was chosen as the default bit string length to use for future experiments and optimization runs on the Brugge case.

4.5.2.2 Population size

13 different population sizes were chosen and a GA optimization was performed for each population size. Crossover rate was set to 0.3. Mutation rate was set to 0.5.

For each generation, the best solution is copied directly to the next generation, as was discussed for low population sizes in section 3.4.4. The population sizes experimented with was: 101, 81, 61, 51, 41, 31, 21, 11, 9, 8, 6, 3 and 2. Since the best individual is kept between generations, number of solutions that are subject to mutation and crossover is really $populationSize - 1$. This means that for population size 2, only 1 individual is mutated per generation, and the other is not counted as a simulation. Since the population sizes vary, the performance should not be measured per generation for comparison, and is measured per simulation instead. As was stated in Goal 3 in section 2.3, the aim is to minimize number of simulations performed. The results of these optimization can be inspected in figure 4.24.

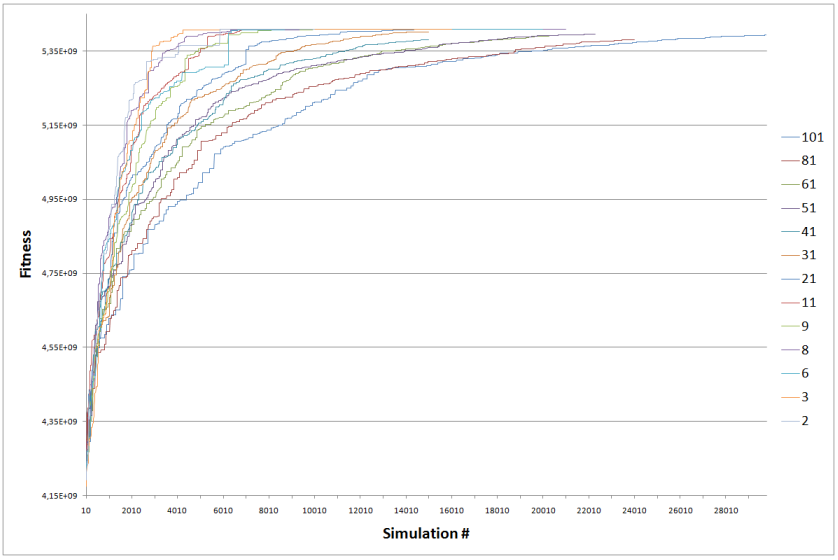

Figure 4.24: Performances for performing optimization with the GA using different population sizes.

We can see that optimization with the largest population sizes improves the solutions a lot slower than optimization using smaller population sizes. Every optimization run with population larger than 11 performs poorly. Figure 4.25 shows the same results but only for the smaller population sizes.

As a result of the variation in population size, the different optimization runs started at slightly different best initial guesses, despite identical random

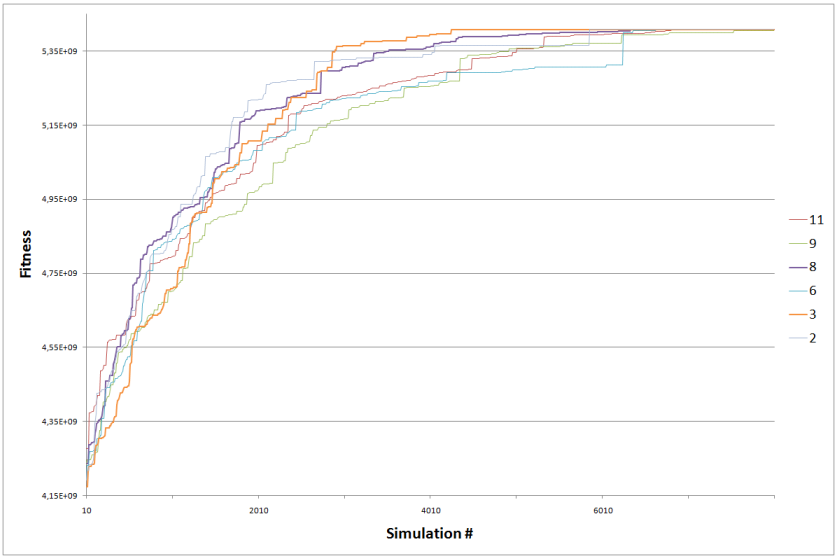

Figure 4.25: Best performances for performing optimization with the GA using different population sizes.

seed. We can see runs with population sizes of 3 and 8 showing promising development of solutions. In the optimization with population size 3, a relatively bad initial guess is assumed, and the GA rapidly improves this solution, eventually finding a good solution and converging on this solution long before all the other optimization runs. Optimization with population size 8 showed very good early improvement of the solutions, and second best convergence to a good solution. The difference in performance between the different low population sizes for optimization is small, and the actual performance differences will be a little random without performing the same experiments several times.

Because the differences are not big, this was not explored further, and population sizes of 3 and 8 were chosen as default for further experimentation and actual optimization runs with the ECLIPSE simulator. The results of scheme 1 in section 4.2 confirms these findings when simulated on the ECLIPSE simulator.

4.5.2.3 Crossover rate

Experiments were performed with 4 different crossover rates for population size 8 and 2 different crossover rates for population size 3. Mutation rate was set to 0.5 in both cases. Figure 4.26 shows the experiments with different

crossover rates for optimization with population size 8. Crossover rates of 0.0, 0.3, 0.7 and 1.0 were used in these experiments.

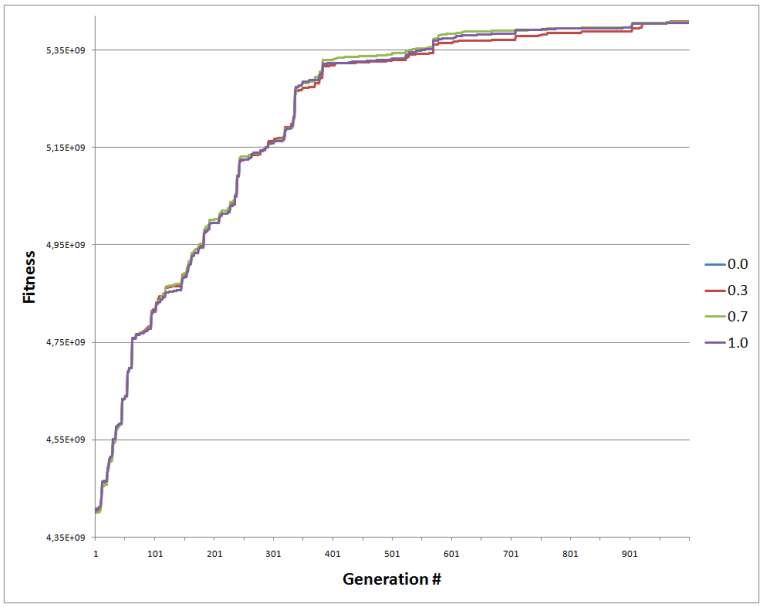

Figure 4.26: Performances for performing optimization with the GA using different crossover rates and population size 8.

We can see this parameter bringing very little performance change to the optimization runs. The higher crossover rates showed slightly faster convergence. Figure 4.27 shows similar results for the optimization run with population size 3. In this case there are only 2 options, either crossover the 2 dynamic individuals in the population, or don't.

The difference is minor, but the higher crossover rate seems to improve slightly faster than the lower crossover rate. Crossover rate of 0.7 was chosen as default for further experiments and optimization runs with population size 8 and 1.0 for population size 3 runs.

4.5.2.4 Mutation rate

Experiments were performed with 7 different mutation rates; 0.0, 0.2, 0.4, 0.6, 0.8, 0.9 and 1.0. Experiments were performed with population size 8 and crossover rate of 0.7. The results of these experiments can be inspected in figure 4.28.

For low mutation rates, the algorithm converges slower than for higher mutation rates. Best values for mutation rates appears to be around 0.8–1.0.

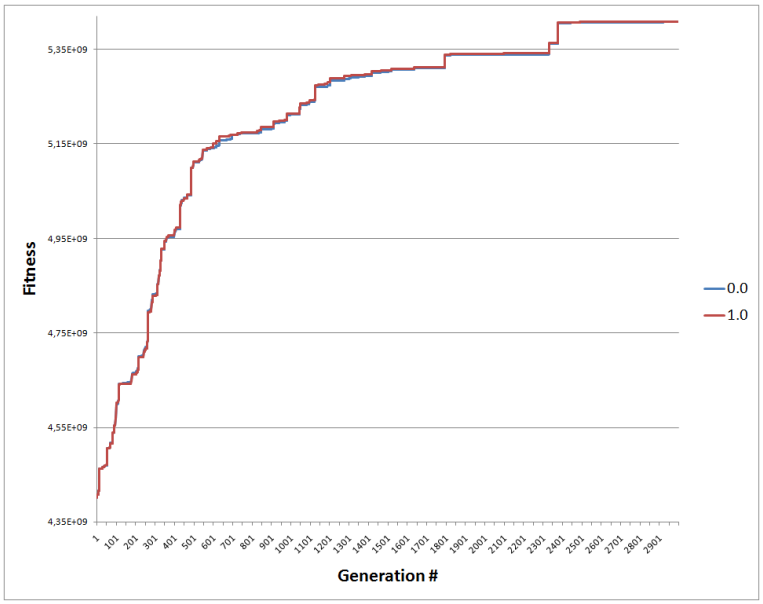

Figure 4.27: Performances for performing optimization with the GA using different crossover rates and population size 3.

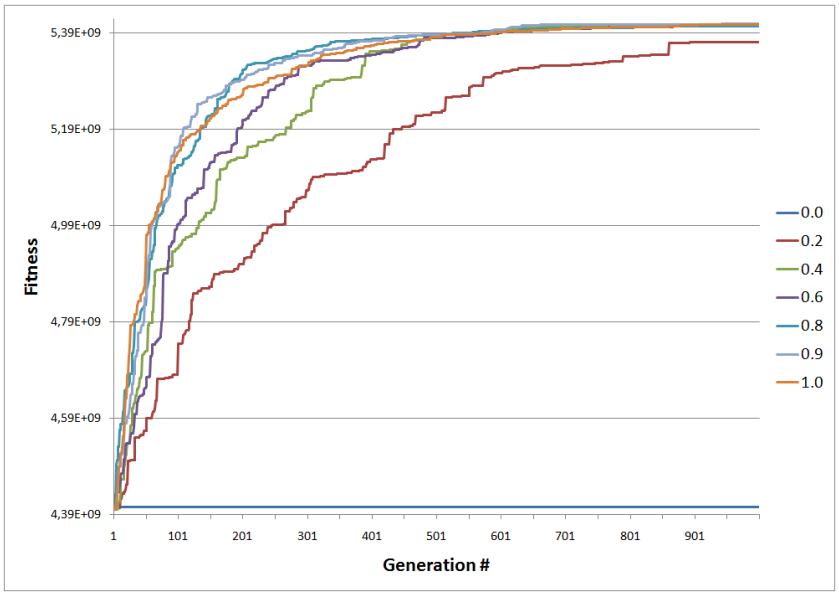

Figure 4.28: Performances for performing optimization with the GA using different mutation rates.

Mutation rate of 0.9 was chosen as the default for optimization runs.

Chapter 5

Comparison of results

In this chapter the optimization results obtained by the different schemes is compared to each other and to other methods for optimization used on the Bugge case. The algorithms presented in this comparison are;

- Scheme 1, GA population size 3 — Presented in section 4.2.
- Scheme 1, GA population size 8 — Presented in section 4.2.
- Scheme 2, 3000 training cases used — Presented in section 4.3.
- Scheme 3, good initial population — Presented in section 4.4.
- Matlab fmincon optimization — Presented in section 2.2.4.1.
- Hooke-Jeeves optimization — Presented in section 2.2.4.2.

Table 5.1 shows some key results from use of the different algorithms. The

			Simulations used to reach:		
Algorithm	Init. guess	Opt. solu.	4.5E+9	4.506e+9	Opt.
Scheme 1, GA3	4383588513	4509211987	430	480	2900
Scheme 1, GA8	4383588513	4511276768	990	1000	3020
Scheme 2, ANN	4020186350	3872832355	NA	NA	3000
Scheme 3	4383588513	4322394899	NA	NA	1911
fmincon	4390801909	4504325848	280	NA	443
Hooke-Jeeves	4384377320	4508690950	300	1200	3000

Table 5.1: Result comparison: The table shows initial guess fitness, optimized solution fitness, number of simulations used to reach fitness values of 4.5E+9 and 4.506E+9, and number of simulations used to reach the optimal solution.

table reflects the efficiency of the algorithms by both evaluating how many simulations were used to reach the optimized fitness value and also how many simulations were used to reach two good solutions. Good solutions are considered by the author to have fitness values above 4.5E+9.

The results can also be inspected in figure 5.1 where the most promising approaches are displayed. These approaches all used a very similar good initial guess, making them very suitable for comparison.

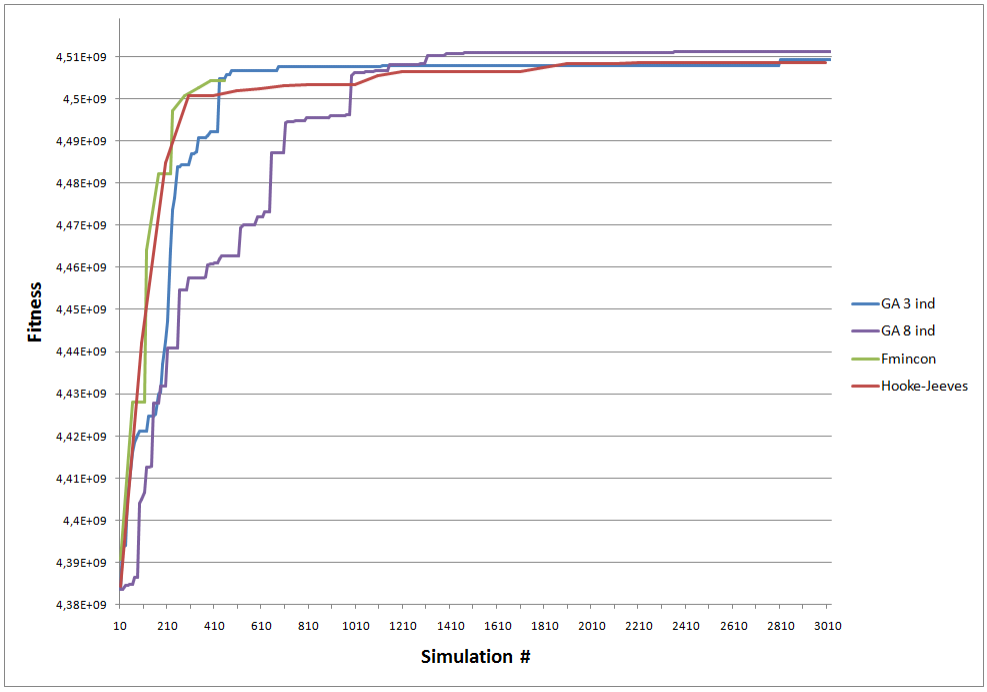

Figure 5.1: Performances for performing optimization on the Brugge case with different optimization algorithms.

The fmincon and Hooke-Jeeves algorithms both provide very rapid early improvement of the solutions, and reach what can be considered a good solution after around 300 simulations. This early performance is not matched by any of the schemes the author has used for production optimization. However, we can see the GAs used by the author in scheme 1 provide the best optimized results if optimization is given the time to run to convergence. The GA using population size of 3 finds a solution after 480 simulations that is not matched by Hooke-Jeeves until after 1200 simulations, and never matched by fmincon. The solutions developed after 1310 simulations by the GA using population size of 8 are not matched by any of the other algorithms. The

optimized solution by this GA is the best result obtained for the Brugge case of all the algorithms in this comparison.

The use of the fitness terminology is not appropriate for all the methods presented. Fitness values and objective function values are both defined by the same NPV calculation given in USD, and thus are perfectly comparable.

Algorithm parameter tuning Section 4.5 illustrated how considerable effort was put into determining good algorithm parameters for the GA and ANN used by the author. Some of the other methods used did not involve the same rigorous tuning of parameters, or information was not present about the work of tuning these parameters. It is possible that some of these algorithms could provide better performance given more tuning of the relevant algorithm parameters.

Fmincon result inaccuracy The results obtained by fmincon for the Brugge case was from a previous study by IRIS that did not use the same formulation of the Brugge case as the one the author has used. They are the result of optimization performed on 10 different formulations of the Brugge case, and the results are the average of all of them. These formulations are all similar to the formulation the author uses, and the results obtained are very nearly the same as the ones the author has used. Lorentzen [11] provides details of the models used.

Thus, the results obtained from fmincon are hard to directly compare. Solution vectors were available from the study using fmincon, and figure 5.2 shows how the optimized solutions from both fmincon and GA optimization compares to each other.

We can see the solution vectors showing similar but not identical properties. When the solution vector discovered by fmincon is run on the author's formulation of the Brugge case the solution is evaluated to 4487549140 NPV. This value is slightly below the value presented in table 5.1. The fmincon results also do not include optimization values beyond the first 443 simulations performed. The differences in the models and approaches used are significant enough to cast some doubt about the veracity of the fmincon results for this comparison.

The findings of this chapter are further discussed in chapter 6.

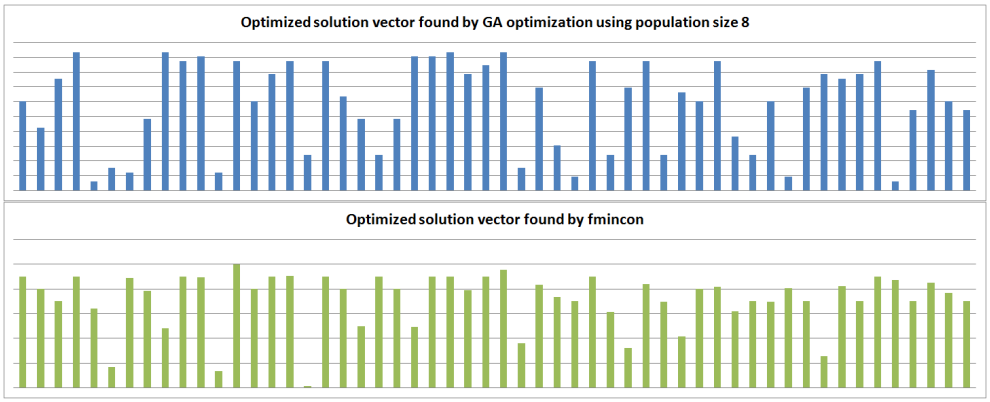

Figure 5.2: Solution vectors for fmincon optimization and GA optimization on the Brugge case. The columns represent optimization parameters in the solutions. The length of these columns represents the value of the corresponding parameter.

Chapter 6

Discussion

6.1 Results discussion

Chapter 5 presented a comparison between the optimization schemes used by the author and other optimization algorithms used on the Brugge case. The results showed the GAs used in scheme 1 provided very good performance in the middle (500+ simulations) and late stages (1000+ simulations) of the optimization runs. The result of perfoming GA optimization with 8 individuals also yielded the best optimized solution of all the algorithms presented. In the early stages of optimization, other algorithms provided rapid early improvements of the solutions that could not be matched by the schemes the author used.

Fmincon (section 2.2.4.1) discovers derivates by intentionally mutating the optimization parameters. These derivatives are used to determine the appropriate direction of change in the optimization parameters. As figure 5.1 showed, this information is very helpful in determining early improvement to the solution. This also means the algorithm has a tendency to get stuck in local maxima with no way of getting out.

Hooke-Jeeves (section 2.2.4.2) is, much like GAs, sometimes referred to as a method to use when other methods fail. It is a pattern search which uses no derivatives. It is however not stochastic. It follows a very specific heuristic to zero in on the maximum (or minimum). Thus it is also subject to getting stuck in local maxima.

GA is a stochastic and derivative-free method, and for low population sizes its behavior can be very random. We can see this clearly in figure 5.1, where both GA approaches improve the solutions sporadically and sometimes in giant leaps. This is the effect of random mutation to the optimization parameters without any specific knowledge of derivatives or similar information

about the effect of mutation on the objective function value. This functionality is instead emergent in the GA through the inherent selection pressure the mechanisms of the GA provides.

The success of the GA in middle and late stages of the optimization shows where the stochastic approach to optimization can provide added value. Random mutation provides exploration of solutions in areas of the search space which can be hard to evaluate by other methods, when they get stuck in local maxima. The ability to get out of local maxima is the main reason for the better results achieved by GA optimization.

It is hard to declare a clear winner among the optimization algorithms. The result obtained after 3020 simulations using the GA is only 0.234% better than the result obtained after 300 simulations using Hooke-Jeeves. When the meaning behind the numbers is inspected, the difference can also be seen as a difference in 10519008 USD for the project. That is over 10 million USD net present value for the cost of performing 2720 simulations.

While the relative difference between the results is very small, the difference in money earned is substantial. This highlights the importance of good optimization for reservoir optimization. Each simulation takes around 90 seconds for the Brugge case. This means 2720 simulations will be performed in just below 3 days. 3 days of simulating for a gain of 10.5 million USD over a 20 year period seems well worth it.

One can easily imagine cases that take substantially more time to simulate than the Brugge case. Simulations can take 10-20 hours. It then becomes very impractical to have to do large number of simulations for optimization.

It is also important to keep in mind that the simulator is only a rough approximation of the real reservoir conditions. Differences on the scale of 0.2% and significantly higher are well within the margin of error for most simulator models of complex reservoirs.

A balance has to be struck between number of simulations and quality of solution according to the specifications of the problem. How long is the simulation time? How significant are small differences in the optimized solutions? Is the simulator considered to be accurate enough for it to matter? These and other similar questions decide the balance, and also decide which algorithm can be considered most appropriate for a given case.

Section 3.1 and figure 3.1 illustrated the modularity of production optimization. The optimizer and simulator can both be replaced. Scheme 3 in this dissertation already introduced the idea of mixing up different methods to be able to provide good results. This idea can be extended to involve combinations of different optimization algorithms.

The results presented in chapter 5 showed that the different algorithms performed differently in different stages of optimization. This indicates that combining the use of the different methods can be done to produce the best results. One could for instance take advantage of fmincon to improve a solution rapidly to a good level in the beginning, and then change to a GA for the middle stages. In the end stage the population size used by the GA could be increased for better exploitation of the discovered solution. Chapter 8 mentions these and other similar ideas for future work with this material.

6.2 ANN discussion

The process of obtaining parameters for training ANNs was presented in section 4.5.1. This section will discuss the findings and highlight interesting properties of the problem and methods used that was discovered during experimentation with the ANN.

6.2.1 Parameter dependencies

The parameters in the backpropagation algorithm are highly dependent on each other. If one changes, the effect of changing another will be different than it would have been before the change in the first parameter. A good example is the relationship between learning rate and number of epochs used. If a high learning rate is used, fewer epochs are needed to reach convergence. If a lower learning rate is used, performing training over more epochs can yield good results. If these factors are not balanced, this can result in unwanted behavior like overtraining the ANN.

This effect can be seen when considering figure 4.19 in section 4.5.1.4. Running training for many epochs resulted in an ANN which performed very well when evaluating the training data. However, the performance when evaluating testing data was worse than for ANNs trained over fewer epochs. Too few epochs used for training resulted in bad performance on both training and testing data sets.

In the experiments in section 4.5.1 the author discovered parameter values for the ANN that proved to perform well in most cases. This was presented in section 4.5.1.5, where the chosen parameters yielded good result for different number of training cases. The chosen parameters can not be guaranteed to be optimal for all cases, but as a result of the testing, they are assumed to be appropriate for most cases.

6.2.2 Low success rate of ANN

Throughout section 4.5.1 performance measurements for the trained ANNs are given, showing successful approximation rate of testing cases to be mostly below 25%. This is assuming correct approximation if the approximated value is within 5% of the actual value. This means that the ANNs are largely unsuccessful at approximating the ECLIPSE simulator. The poor results presented for scheme 2 when used to perform optimization on the Brugge case in section 4.3 confirms this finding. In all cases the optimization produced solutions worse than the initial guess for scheme 2.

Table 4.2 showed large differences between the approximated and actual fitness values of the optimized solutions. The table also showed that the fitness value of the initial guess was very accurately approximated by all the ANNs used. This behavior was also seen for the results of scheme 3 in section 4.4. When the optimization scheme approximated solutions outside the explored solution space the approximation was very inaccurate, while for solutions previously explored by ECLIPSE the ANN approximation was surprisingly accurate. Two plausible explanations exist for this.

For scheme 2, the initial guess could be part of the training data sets for all cases. The author has made certain the exact solution was not part of any of the training data sets, and it is highly unlikely that a very similar solution was part of all the training data sets.

The other good explanation comes from inspecting how the optimization parameters in the Brugge case works. Section 3.2.2 discussed how oil production is controlled in the well segments, and figure 3.9 shows how the different segments are turned off when producing too much water compared to oil. For many of the producer wells, the values for the top segments are less influential than the bottom ones. Some of them can in many cases produce for the entire production horizon without ever starting to produce water. The maximum allowed water cuts can then be any value for these segments without changing the total outcome. This likely changes for optimal production parameters, where the total "bandwidth" of the system is tuned by finding good values where the top segments in the producers are exposed to water.

The effect is that very many combinations of parameter settings in the solution space yields production values around a certain value range. Only a few optimal combinations of parameters change the outcome significantly. The evaluation of the training data presented in section 4.5.1.6 confirms this. Figure 4.22 showed that 85.05% of the training data set evaluated to fitness

values between 3.65E+9 and 4.05E+9. When 85.05% of the possible solutions in the solution space evaluates to values in a certain range with low individual difference, it follows that the chances of randomly generated training data falling inside this category of solutions is larger than for significantly better or worse solutions.

The initial guess for scheme 2, with a value of 4.02E+9, falls inside this value range. The ANN is trained with cases that focus on values in the given range. To correctly approximate the initial guess is therefore more likely than correctly approximating the optimized value. Only 7.23% of the solutions in the training data set was above the 4.05E+9 value, and when we inspect figure 4.21 we see that none are close to the value that was found to be optimal by scheme 1, 4.51E+9. **The ANN is a lot more successful at approximating solutions that are mediocre, than at determining which ones are significantly better or worse than the average.**

Section 4.1.2 and 4.1.3 presented proof of concepts for scheme 2 and 3 on the shoebox model. Use of ANNs on this model did provide much better results than on the Brugge case. For the shoebox case, the best optimization results was found using scheme 3, where ANNs used training data developed during the optimization to approximate the ECLIPSE model. These experiments featured only 16 optimization parameters to optimize, while the larger Brugge case involved 54 optimization parameters to optimize over. The shoebox case also has a far simpler internal structure, containing 64 cells, where the Brugge case contains over 60000.

It follows that the optimization parameters for the Brugge case allows for a larger number of possible input–output combinations. The degrees of freedom in the optimization variables are simply too many to allow for accurate approximation with ANNs. When we view the results of scheme 3 for both the shoebox case in section 4.1.3 and the Brugge case in section 4.4 together, we can see clearly the problem that arises from too many degrees of freedom.

The ANN provides accuracy in the areas which has been explored by the GA and ECLIPSE simulator. The ANN is used for optimization, resulting in a solution that is outside the explored optimization parameter space. Solutions in this area of the solution space is explored and added to the training data. The new training data is then used for training a new ANN, and a different area is located that needs to be explored. This cycle was repeated around 5 times in figure 4.4 for the shoebox case until the ANN approximation of the ECLIPSE simulator was good enough to provide good optimization results using the ANN. This effect was illustrated in figure 3.5 in section 3.1.3. For the Brugge case, figure 4.13 shows the same develop-

ment. Because of the larger number of possible solutions for the Brugge case, the algorithm does not converge to one solution with nearly the same performance as for the shoebox case. It does not converge to one such solution at all in the given results.

It is possible that given enough time, scheme 3 would yield good results also for the Brugge case. This would likely happen when a much larger number of significant areas of the solution space had been explored, and there is no way to know exactly when this would occur.

6.2.3 Random and incremental training data

The idea behind the incremental way of producing training data presented in section 3.5.6 was to distribute solutions evenly across the solution space to provide training data for as many as possible areas of the solution space. Section 4.5.1.6 showed that both the incremental and the randomly generated training data sets produced solutions with fitness values distributed similarly. ANNs trained using incrementally generated training data for scheme 2 in section 4.3 approximated the optimized solutions worse than the ANNs using randomly generated training data. When we consider the findings in figure 4.22, we see that the randomly generated training data contains slightly more cases with evaluated fitness values above 4.05E+9 than the incrementally generated training data.

It follows that the randomly generated training data should be slightly better able to approximate solutions in the upper end of the fitness spectrum. Since the focus of optimization is on this end of the solution space, the slight advantage the randomly generated training data gives is large enough to be observable, as it was for scheme 2.

6.2.4 Choice of structure for the ANN

In section 4.5.1.2, using two hidden layers was shown to be producing the best result. The structure used when training ANNs consisted of 54 nodes in the input layer corresponding to each input variable in the Brugge model and 1 node in the output layer also corresponding to the one value returned when simulating the Brugge model on the ECLIPSE simulator. Two hidden layers were used between the input and output layers, the first consisting of 100 nodes, the second containing 50 nodes. The effect of two hidden layers is described by Mitchell [15]; “Any function can be approximated to arbitrary accuracy by a network of three layers of units... the output layer uses linear unit, the two hidden layers use sigmoid units, and the number of

units required at each layer is not known in general". This is the structure the author has chosen.

The results presented in section 4.5.1.2 helped determine the number of nodes for the hidden layers. The discussion that follows is for the training using random training data, as this was seen to give best performance. When using few nodes in the hidden layers, the influence of each input node was decreased, which helped combat noise and overfitting. When more nodes were used in the hidden layer, the inputs were given more influence which prevented underfitting. These approaches each have their pitfalls. The effects can be either falsely removing too much noise or allowing insignificant inputs too much influence on the output value.

These effects can be observed when considering figures 4.16 and 4.17 together. In figure 4.17 we see the training using a single layer of 64 nodes converging to lower error in the output layer than the training using 100+50 nodes. The low error means that the trained ANN is good at approximating the training data. Despite the higher error, the actual performance of ANNs on testing data in figure 4.16 paints a different picure. The ANN trained with 100+50 nodes scored better when used to simulate the testing cases. The ANN using fewer nodes offers a lower level of complexity in the hidden layer, giving inputs falsely large influence over the outcome, and making the ANN perform poorly on unknown cases.

The performances displayed in figure 4.16 showed that a balance between using too few and too many nodes was struck using the chosen structure of 100+50 nodes in the hidden layers.

6.3 GA discussion

The process of obtaining parameters for running GA optimization was presented in section 4.5.2. This section will discuss the findings and highlight interesting properties of the problem and methods used that was discovered during experimentation with the GA.

6.3.1 Search space topology and parallel search

In the experiments to determine good population sizes for performing optimization using the GA, low population sizes were found to give superior results over larger population sizes. Since the population size can be viewed as the degree of parallel search offered by the GA, it can be assumed that this problem does not take advantage of the available parallelism. Parallel search is great for avoiding getting stuck in local maxima, by exploring mul-

tiple promising solutions at the same time. As figure 4.24 in section 4.5.2.2 showed, all the optimization runs were able to converge to approximately the same solution without getting stuck in local maxima. This indicates that a global maximum can be found, and likely from many different directions in the search space. When the search space contains few or no local maxima, the algorithm should not waste time exploring alternative solutions, but can instead focus on improving one promising solution as fast as possible. This is an effect of low population sizes for optimization, and the reason for their success on this problem.

These assumptions were put to the test in section 4.2 for scheme 1 on the ECLIPSE simulator. Figures 4.7 and 4.8 showed that GA optimization using 3 individuals did converge on a good solution faster than the optimization runs using larger population sizes. Figure 4.9 showed the different optimized solution vectors for scheme 1. When we inspect the similarities between the optimized results we see that a few common features are present for all 3 variations. Section 6.2.2 discussed how some segments in the producer wells were less relevant than others for the total outcome, and this is very visible in the figure. This means that the GA optimization using 3 individuals is not necessarily stuck in a local maximum. The chance of a random change to the parameters resulting in a change in a significant part of the parameter space is low. The GAs using higher population sizes offer a larger degree of exploitation of good solutions. Although they reach good solutions slower, they are better able to make small changes to existing good solutions. As figure 4.8 showed, the GA optimization runs using larger population sizes were able to produce better solutions in the long run.

6.3.2 Solution quality versus number of simulations

Goal 3 in section 2.3 emphasized the importance of minimizing number of simulations used for any of this work to be relevant from the industry point of view. The results of scheme 1 was presented in section 4.2. These results showed that minimizing the number of simulations could come at the cost of quality of the solution. This was illustrated by pointing out that the 0.123% difference between the result obtained after 460 simulations and the result obtained after 3030 simulations is actually a difference of 5.5 million USD in the NPV calculation. The balance between these factors were discussed in general in section 6.1, and the same observations are relevant when considering the balance between these factors for different GAs.

6.3.3 Population fitness variance

Low population sizes and high mutation and crossover rates were determined to perform very well on the optimization runs. This means that for each generation, a lot of change is introduced in a small population. Since one individual is always copied directly between generations, a best solution will never be lost. It then makes sense to make major changes to the remaining population in hopes of establishing a better solution. One effect is that population fitness variance becomes a very unreliable measure of convergence. Random changes can make a population very varied in one generation, while being almost completely homogenous the next. It follows that population fitness variance was not considered for evaluating convergence in GA optimization with low population sizes.

6.3.4 The impact of genetic operators

The individuals in the population of one generation are based on the best individuals of the previous generation. For small population sizes, the same individual often serves as the basis for the majority of the future population, especially if this solution is superior to all the others. This means that the crossover genetic operator often works on identical solutions. And when you cross two identical solutions with each other the same solutions becomes the end result. This effect was observed in figures 4.26 and 4.27 in the presentation of crossover rate experiments in section 4.5.2.2. The crossover operator did not have a major impact on the success of the optimization runs. For population size 8 the crossover rate was seen to have a bigger impact than on the experiment with population size 3. If larger population sizes had proved more efficient, this GA parameter would have had an even bigger impact.

The changes to the population introduced by the mutation operator proved to be the driving force in improving the solutions.

Chapter 7

Conclusion

Through the research performed in this dissertation the author has gained valuable experience and insight into practical engineering use for methods from the AI domain. The optimization schemes discussed in this dissertation produced results that were promising in comparison to other relevant methods for use in the industry. The point of performing optimization on oil reservoir models is ultimately to maximize the profit from production.

7.1 Genetic algorithms for optimization

Section 6.1 discussed how the GA used by scheme 1 could produce the best results with a slightly higher number of simulations used than the other methods in the comparison. The better results were shown to be a significant increase in profit even if their relative difference was not as significant.

The cost of performing thorough optimization can be seen as very low when compared to the consequences to profit with unoptimized production (section 6.1). In that regard, the methods developed in this dissertation could be seen as a revolution. This is however not the case since the models used to simulate production are not accurate enough to capture the full reality of sub-sea oil reservoirs. Optimization is generally performed on a rougher scale than for the minor differences between optimization results that was presented in this dissertation.

Seen in this light, the results are more interesting as a comparison of GAs as derivative-free, stochastic optimizers and other methods using traditional hillclimbing techniques. The results were as could be expected. The GA would find initial good solutions slower than the SQP method. Where other methods can get stuck in local maxima, the stochastic exploration of the solution space provided mechanisms to escape local maxima, and yielded

better results in the long run.

While the industrial relevance of the good results from this research is questionable now, one can imagine that simulating tools will see improvement in the future. If the precision of simulating tools increases the methods developed in this dissertation can be seen as more effectual also from an industry point of view.

7.2 Artificial neural networks used in optimization

Not all technologies developed in this dissertation lived up to the expectations of the author. Scheme 2 and scheme 3 were developed as alternative methods of performing production optimization. These schemes showed great promise in initial testing and for the simpler case studied (section 4.1.3). When optimization was performed according to these schemes on the larger case, they were seen to not significantly increase — and even decrease — the performance of the solutions they were trying to improve (chapter 5). The ANNs used were not able to capture the full complexity of the Brugge case. This result yielded valuable insight into what ANNs can not be expected to do.

The Brugge case is a model with significant internal complexity and many degrees of freedom. The smaller shoebox case proved easier to approximate. Classification is a typical application area for backpropagation ANNs like the ones the author used. This is where ANNs perform well, as a means of simplifying the underlying concepts in a system. When the author attempted to use ANNs to simplify the Brugge model, the received result was as could be expected, a simplification. When this simplification was directly compared to the real model, discrepancies between the different approaches were inevitable.

The ANN was able to produce good results for the simpler shoebox model and for parts of the solution space of the Brugge model. This is in reality a success for the use of ANNs. It shows that the ANN performed its task of simplifying the problem. The author entered into this research with unrealistic expectations of what the ANN was supposed to do, and was through the research made aware of the limitations of the technology.

7.3 Scheme 3 ideas

The ideas in scheme 3 were intended to provide a genuine contribution to both the AI field as well as for industry purposes. The incremental approach to training ANNs proved to be inefficient as a means of augmenting optimization on the Brugge model. It is however conceivable that the concepts and ideas developed can be successfully employed for other challenges, both in the petroleum industry and other fields. The best indication of this can be seen when performing scheme 3 optimization on the shoebox case.

Section 4.1.3 discussed the results of scheme 3 applied to the shoebox case, and these results showed that incrementally generating training data for the ANNs according to scheme 3 can work well. Scheme 3 produced an optimized solution for the shoebox model in nearly half the number of simulations required for the GA of scheme 1 to achieve the same result (section 4.1.4).

7.4 Multidisciplinary focus and the AI methods used

This dissertation involves topics from many different disciplines, and considerable effort has gone into identifying and understanding the challenges in domains other than AI. The assistance of petroleum and process engineers and engineering students was crucial for working with the topics covered in this dissertation. As a result of this multidisciplinary focus, the AI methods studied in this thesis have been limited to standard and straightforward implementations of methods from the AI domain. While the success of backpropagation/feedforward ANNs and basic GAs is undeniable, for this dissertation and for other applications, more advanced approaches exists. If this study was concerned only with topics and methods from the AI domain, it would be appropriate to investigate other variations of the methods for performance comparison.

7.5 Goals

Section 2.3 discussed the goals of this dissertation. The following subsections will summarize how these goals were met.

7.5.1 Goal 1: Use of GA

Section 6.1 discussed the performance of the approaches used by the author to perform production optimization on the Brugge model. The comparison showed that the use of GA for production optimization proved effective, especially in middle and late stages of optimization. The usability of the method then depends on specifics of the problem, like the cost associated with evaluating solutions. For both the complex Brugge case and the less complex shoebox case GAs were shown to be competitive algorithms that can be used successfully for reservoir production optimization.

7.5.2 Goal 2: Use of ANN

Section 6.2.2 discussed the low success of ANNs and pointed out how there are too many degrees of freedom to the Brugge case to be successfully approximated by ANNs. The ECLIPSE model offers a complexity that the ANNs used by the author were not capable of capturing. While the use of GAs proved to be effective at solving the problem given by the Brugge case, the use of ANNs according to schemes 2 and 3 did not yield the same good results. Looking back at sections 4.1.2 and 4.1.3, we saw that these schemes proved a lot more successful for optimization on the less complex shoebox case. ANNs can be a very appropriate technology for use in less complex cases where the number of parameters and complexity is more manageable.

7.5.3 Goal 3: Minimizing simulations used versus quality of solution

Chapter 5 presented performance comparison for the different schemes used, and section 6.1 discussed these findings. It was pointed out that the GAs used did not provide the best performance in the early stages of the optimization runs, but could provide very good results in the middle and later stages of optimization. As has been thoroughly discussed, a balance should be struck between number of simulations and quality of solutions. Work with the optimization problems in this dissertation has constantly been focused around this issue, and the author therefore considers this goal to be successfully met.

Chapter 8

Future work

This chapter will discuss work that could be done with the material in this dissertation that the author did not get to do because of the time limitations for the project.

8.1 Scheme 3 expanded

The ideas put forth in scheme 3 (section 3.1.3) involved using multiple different approaches for evaluating solutions in a single optimization run. This is possible through the modularity of optimization seen in figure 3.1 in section 3.1. The same modularity applies to the optimization algorithm employed.

The results in this dissertation show how different algorithms and approaches have their individual advantages and disadvantages for reservoir optimization, often relating to accuracy versus computing cost. Some methods are better for rapid early convergence, while others are better at late exploration for the global maximum. In the same manner as solution evaluation mechanisms were interchangeable in scheme 3, a method can be developed that also alternates between different optimization algorithms.

A method could be developed that used SQP in the initial phases of optimization. The method could switch to using a GA with low population size for the middle stages, and use a GA with higher population size in the late stages of optimization. Such a method would involve the best from multiple methods, and would probably offer performance better than for any of the methods on their own.

This could be further augmented by also alternating the evaluation methods. This does not need to be limited to only the simulator and the ANN. Other simplification and approximations of the reservoir conditions could also be used.

8.2 Parallel computing

The independent evaluation of fitness in GA populations makes the algorithm very suitable for use in parallel computing. This has not been discussed in this dissertation because the ECLIPSE simulator already offers parallel functionality. It is conceivable that the parallelism offered by the ECLIPSE simulator is less efficient than the parallelism offered by GAs. Experiments could be performed to compare performance of the parallelism in both technologies.

A parallel GA used on ECLIPSE running in serial mode could be compared to a serial GA used on ECLIPSE in parallel mode. We assume the scheme using parallel GA can be shown to perform better than the scheme using parallel ECLIPSE. GAs as optimization algorithms in a distributed computation scheme would then perform even better in comparison to other methods incapable of performing evaluations in parallel.

8.3 Training ANNs with the history data used to develop simulator models

Figure 3.1 shows how the ECLIPSE simulator models are made using data from the actual performance of a field. Other information about the geological properties of the reservoir can also be used. One can imagine using this data directly to train ANNs to model reservoir behavior instead of going through the ECLIPSE simulator models for generating training data. ANNs could be trained using the exact same data used to train the original simulator models.

Time limitations were not the only reason for the author not investigating this possibility. It requires a lot of domain knowledge about the geological and fluidic properties of such reservoirs, as well as vast knowledge of the workings of oil wells and oil production facilities in general. The author does not possess this knowledge through studying informatics and AI. The availability of such history data is also problematic. The author has not been working on reservoirs that exist in the real world. This kind of information is not readily available for anyone to use, and the author did not have access to such data for this thesis. The data used to develop simulator models can also include image data of the geology of the reservoir and similar information that is hard to directly put into an ANN.

To perform the proposed training of ANNs would likely require a multidisciplinary effort as well as available real history data and geological information for an oil field.

List of Figures

List of Tables

Bibliography

[1] Masoud Asadollahi et al. Production optimization using derivative free methods applied to brugge field. *In preparation*, 2009.

[2] Jan Bjålie et al. *Menneskekroppen: Fysiologi og anatomi*. Gyldendal Norsk Forlag, first edition, 2006. Chapter 2, p. 55–97.

[3] Rober Callan. *The essence of neural networks*. Prentice Hall Europe, first edition, 1999.

[4] L.P. Dake. *Fundamentals of Reservoir Engineering*. Elsevier, 1998.

[5] Charles Darwin. *On the Origin of Species*. Dover Publications, Incorporated, 2006.

[6] David Edward Goldberg. *The design of innovation*. Springer, second edition, 2002.

[7] John H. Holland. *Adaption in natural and artificial systems*. MIT Press, 1992.

[8] Kenneth A. De Jong. *Evolutionary computation: a unified approach*. MIT Press, 2006.

[9] Eric R. Kandel et al. *Principles of neural science*. McGraw-Hill, fourth edition, 2000.

[10] S. Kirkpatrick et al. Optimization by simulated annealing. *Science, New Series*, 220(4598):671–680, 1983.

[11] Rolf J. Lorentzen et al. Closed loop reservoir management using the ensemble kalman filter and sequential quadratic programming. *2009 SPE Reservoir Simulation Symposium*, 2009.

[12] George F. Luger. *Artificial Intelligence*. Pearson Education Limited, fifth edition, 2005.

[13] Thomas Malthus. *An Essay on the Principle of Population*. Kessinger Publishing, 2004.

[14] The MathWorks Inc. *Matlab R2008b (r)*. http://www.mathworks.com, 1984–2008.

[15] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[16] Michael Nikolaou. An overview of industrial model predictive control technology. *Advances in Chemical Engineering Series, Academic Press*, 1998.

[17] Jorge Nocedal et al. *Numerical Optimization*. Springer, second edition, 2000.

[18] Statistics Norway. Oil and gas. *Avaliable: http://www.ssb.no/emner/10/06/olje_gass/*, 2008.

[19] Young Whan Park et al. Process modeling and parameter optimization using neural network and genetic algorithms for aluminum laser welding automation. *The International Journal of Advanced Manufacturing Technology*, 37:1014–1021, 2008.

[20] David M. Prett et al. *The Shell Process Control Workshop*. Butterworths, 1987.

[21] David M. Prett et al. *The Second Shell Process Control Workshop: solutions to the Shell standard control problem*. Butterworths, 1990.

[22] J. Qin et al. An overview of industrial model predictive control technology. *Fifth International Conference on Chemical Process Control*, 1997.

[23] S.S. Rao. *Engineering optimization*. Wiley-IEEE, third edition, 1996.

[24] Mark Ridley. *Evolution*. Wiley-Blackwell, third edition, 2004.

[25] Schlumberger. *ECLIPSE 2008.1*. http://www.slb.com/content/services/-software/reseng/index.asp?, 1982–2008.

[26] William M. Spears. Crossover or mutation? *Foundations of Genetic Algorithms*, 2:221–237, 1992.

[27] Oswaldo Velez-Langs. Genetic algorithms in oil industry: An overview. *Journal of Petroleum Science and Engineering*, 47:15–22, 2005.

[28] D. Yang et al. Integrated optimization and control of the production-injection operation systems for hydrocarbon reservoirs. *Pet. Sci. Eng*, pages 69–81, 2003.

[29] Maarten Johan Zandvliet. Model-based lifecycle optimization of well locations and production settings in petroleum reservoirs. *Technische Universiteit Delft (http://repository.tudelft.nl/file/946615/379065)*, 2008.