NTNU

Norwegian University of
Science and Technology

Study of Pattern Search Optimization and Implementation of Hooke-Jeeves Direct Search for Production Optimization using Perforations

Silje Møller

Master of Science in Engineering and ICT
Submission date: June 2016
Supervisor: Jon Kleppe, IPT
Co-supervisor: Mathias Bellout, IPT
Einar Bauman, IPT

Norwegian University of Science and Technology
Department of Petroleum Engineering and Applied Geophysics

Abstract

This thesis concerns the topic of petroleum production optimization using perforations. A plug-in by PhD student Einar Baumann was extended to include a second pattern search method: Hooke-Jeeves direct search. Originally only the compass search method was implemented.

The thesis covers the topic of these two pattern search methods and how they are applied to optimize well perforation placement for a given objective function value. We present descriptions of how the existing plug-in was modified to make room for a second algorithm, and how the Hooke-Jeeves direct search was implemented. This thesis also presents the well perforation placement optimization problem. The plug-in only optimize for perforation placement, but can in the future be extended to optimize with other algorithms, or for other well completion types, e.g. ICDs.

Two experiments are completed to analyze the behaviour of the two algorithms. The first experiment concerns the Hooke-Jeeves direct search, and how it reacts to changes in the input parameters. In the second experiment we compare the two pattern search algorithms when conducted with the same input parameters. The experiments show that two parameters, initial step length and segment resolution, are dependent on the simulation case. We also find that Hooke-Jeeves direct search is a more efficient pattern search method than compass search when it comes to total number of simulations.

Sammendrag

Denne masteroppgaven angår optimering av perforeringsplassering ved petroleumsproduksjon. En eksisterende plug-in av PhD student Einar Baumann har blitt utvidet med en ekstra *pattern search* algoritme: *Hooke-Jeeves direct search*. Originalt var bare *compass search*-algoritmen implementert.

Masteroppgaven presenterer teori om disse to algoritmene og forklarer hvordan de blir brukt til å optimere plasseringen av brønnperforeringer for en gitt objektiv funksjon. Vi presenterer beskrivelser av hvordan den originale plug-inen ble modifisert for å kunne implementere den andre algoritmen, og om hvordan *Hooke-Jeeves direct search*-algoritmen ble implementert. Oppgaven definerer også problemet som omhandler plassering av brønnperforeringer. Den nåværende plug-inen optimerer kun for plassering av brønnperforeringer, man kan bli utvidet til andre brønnkompletteringer i fremtiden, f.eks. ICDer.

To eksperimenter ble gjennomført for å analysere hvordan de to algoritmene oppfører seg. Det første eksperimentet ser på *Hoooke-Jeeves direct search*-algoritmen og studerer hvordan den reagerer på endring av de ulike parameterne. I det andre eksperimentet ønsker vi å sammenligne de to algoritmene når de bruker de samme parameter verdiene. Eksperimentene viser at initiell steglengde og segmentoppløsning er avhenging av simulerings-casen vi bruker. Resultatene viser også at *Hooke-Jeeves direct search*-algoritmen er mer effektiv enn *compass search*-algorithmen når det kommer til totalt antall simuleringer.

Preface

This thesis is written as a part of the Master's degree in Engineering and ICT with specialization in Integrated Operations in the Petroleum Industry, at the Department of Petroleum Engineering and Applied Geophysics at the Norwegian University of Science and Technology, NTNU. It was written in the spring of 2016 under the supervision of Prof. Jon Kleppe, and co-supervised by Postdoc. Mathias Bellout and PhD student Einar Baumann. This thesis is written in collaboration with the Petroleum Cybernetics Group at NTNU.

We assume the reader is familiar with basic programming.

Acknowledgements

I want to thank my supervisor Jon Kleppe for the opportunity to work with the Petroleum Cybernetics Group at NTNU and for his feedback during this thesis.

Also, I would like to thank Mathias Bellout for our conversations regarding optimization, and for showing me the benefits of applying it.

I highly appreciate the work done by Einar Baumann in the original Optimization plug-in. The structure of his implementation was very helpful for me when I was to extend the plug-in with a second algorithm. His expertise spans many topics, and I would like to thank him for always trying to answer my big and small questions.

At last, I would like to thank my family for their support through the course of my Master's degree.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Reforming the petroleum workflows The recent years downfall of the oil price [1] have forced the petroleum industry to cut costs and make processes more efficient. The petroleum industry has been applying optimization for many years already, but we will probably see even more interest for this topic in the upcoming years. In the conference review [2] of the International Petroleum Technology Conference (IPTC) in December 2014, we find arguments to why petroleum production optimization is important during recession periods. Some of the people attending the conference have experienced this before, and they argue that the industry needs to adapt themselves for these cycles. Instead of reacting with panic they should use the recession as an opportunity to improve their workflows and make them more efficient. The petroleum industry does not want the cutting of costs to sacrifice the quality of the work. A topic in the conference was to draw lessons from projects in the US shale. Major increase in unconventional wells in the US demonstrate that there are other ways to solve problems. Trying and failing with the unconventional wells have resulted in good new solutions which are also applicable for the conventional segment. In the conference, Bernard Montaron, the technical director for Asia at Schlumberger elaborated on the US shales. He said that,

> " Almost 50 % of the wells drilled between 2010 and 2013 are uneconomic either because they were drilled in wrong place, or landed in wrong zone or poorly completed, ... (Approximately) 40 % of perforations in the US shale don't produce. "

He argued that the industry must improve their work in all levels of the chain. The lesson we can take from the US shale is to instead of just drilling anywhere in search for good results, we can spend more time analyzing and preparing the tasks ahead so that projects can be utilized to their fullest. Expanding the preliminary phases to include more optimization studies will benefit projects in the future, especially in times of tight economy. Recession may also lead businesses to apply optimization

to ongoing projects instead of pursuing new, possibly expensive projects. Bernard Montaron's last quote in the conference review sums it up:

> " True differentiation is in maximizing well productivity through optimum use of technology, and that can have much greater and long-lasting impact on economics. "

Applying optimization workflows to the planning and execution of petroleum production will benefit the industry not only now, but also in the next downturn in the oil market.

Optimization problem In this thesis we study the use of pattern search methods for optimizing the placement of perforations along a well. Pattern search methods are a type of derivative-free optimization. The two pattern search methods discussed in this thesis is compass search and Hooke-Jeeves direct search. These two algorithms produce perforation configuration candidates, in search for the perforation configuration resulting in the best objective function value.

Software The software being used in this thesis is mainly products from Schlumberger. The Petrel E&P software is the platform for our work. We implement a second optimization algorithm to an existing Petrel plug-in by using the Ocean Software Framework. The Ocean framework is a class library using the object-oriented programming language C#. With the Ocean framework we can run reservoir simulations using the ECLIPSE simulator on the Petrel platform.

Ocean plug-in The implementation work completed during this thesis is the extension of an existing Ocean plug-in by PhD student Einar Baumann. He has implemented the compass search method to solve the perforation placement problem. This thesis concerns the implementation of a second pattern search method: Hooke-Jeeves direct search. As a consequence of implementing the second algorithm, modifications to the existing plug-in was completed.

Organization of this thesis The next chapter describes the optimization theory being applied in this thesis. Chapter 3 explains the problem concerning perforation placement and why we use pattern search methods. The implementation of Hooke-Jeeves direct search into the existing Ocean plug-in is described in Chapter 4. In chapter 5, we conduct experiments where we apply the algorithms implemented in the plug-in. The experiments study how Hooke-Jeeves direct search behaves when changing the input parameters in the algorithm, and we compare the two algorithms when solving identical cases. At the end we summarize the work that iscompleted and elaborate on some possible further work.

Chapter 2

Pattern search optimization

This chapter defines the optimization theory being studied and applied in this thesis. The chapter starts with an introduction of general optimization theory, before going into derivative-free optimization. Afterwards we describe pattern search methods, which is a sub-category of derivative-free optimization. At last we go into the details of two pattern search methods: *compass search* and *Hooke-Jeeves direct search.*

2.1 General optimization theory

Numerical optimization is a wide topic and can be elaborated through many pages. In this thesis we have reduced the scope to focus on pattern search methods. The following in this chapter will try to give a structure for the optimization theory that is relevant for this thesis. We want to give the reader an overview of where to place pattern search methods within the optimization theory.

2.1.1 Optimization theory concepts

Before going into the details of the algorithms, some general optimization theory concepts is explained. A general definition of the optimization procedure is given below, followed by a description of an objective function, control variables, and constraints.

> " Mathematically speaking, optimization is the minimization or maximization of a function subject to constraints on its variables. [3] "

Objective function is the value that an optimization run want to either minimize or maximize. In petroleum production optimization it is common to use cumulative oil production, net present value, etc.

Control variables are parameters that are altered in the process of finding the maximum or minimum of the objective function. There is a wide range of possible control variables for petroleum production, and it all depends on the target of the optimization. Examples of a control variable for petroleum production optimization can be the injection rate of water or the placement of a well completion.

Constraints are equations and inequalities that must be fulfilled by the control variables before the objective function can be calculated. If all the constraints are satisfied, the objective function will be calculated at a feasible point [3]. In some cases, selecting the constraints will be very intuitive because of the nature of the problem. Other cases will require more detailed constraints if the problem is complex. If the control variable is injection rate of water, a maximum injection rate can be set as the constraint. The beginning and end of a well segment can be used as constraints for a valid placement of a well completion.

2.1.2 Numerical optimization categories

Within numerical optimization we have a category of algorithms called derivative-free methods. A subcategory of derivative-free methods is pattern search methods. In this thesis we are concerned with two pattern search algorithms: *compass search* and *Hooke-Jeeves direct search*. In literature the terms derivative-free, pattern search and direct search are sometimes used interchangeably, but this thesis will use the structure and notation presented in this chapter. The following sections will explain the different categories and describe the two algorithms that is being applied in this thesis.

2.2 Derivative-free optimization

Derivative-free optimization is applicable when the derivative of our objective function is difficult to calculate, or when it is not available. This is often the case when engineers try to solve practical problems using simulator software [3]. Derivative-free optimization is considered relatively easy to implement and are therefore widely used as the first approach.

When discussing derivative-free optimization, it is common to limit the optimization problem to the unconstrained case

$$\min_{x \to \infty} f(x), \tag{2.1}$$

where x is the variables of the objective function f, and the solution space have n dimensions where each dimension represents one of the variables [3]. We use the unconstrained case for derivative-free methods because constraint handling in these methods is still being explored by researchers. It is therefore easier to use the unconstrained problem when discussing derivative-free methods [3].

In this thesis the focus has been on pattern search methods, which is the next topic. Other algorithms within derivative-free methods are the Nelder-Mead simplex method, conjugate-direction methods and simulated annealing [3].

2.3 Pattern search methods

The name "pattern search" was first introduced by Hooke and Jeeves [4] in a paper from 1961. Almost all literature on this topic refer to Hooke and Jeeves and the theory they present in their paper. The term pattern search has since been used as a collective name for all methods that search from a current point in a predetermined direction for a better function value [3]. If the search finds a better point, this point becomes the new base and the search restarts from this point. If a search is not a success, the search direction is changed, or the search perimeter is cut down by reducing the step length.

Different pattern search methods have been developed over the years and adjusted for the optimization problems that they attempt to solve. Pattern search methods tends to have a very simple tactic and therefore easy to use as the first optimization approach. In addition to their simplicity, the pattern search methods are also flexible and reliable [5]. The following sections will present two pattern search methods: *compass search* and *Hooke-Jeeves direct search*.

2.4 Compass Search

The compass search method is, as indicated by its name, inspired by the compass directions. Compass search is also known as coordinate search, local variation, alternating directions, axial relaxation, and alternating variables [6]. The algorithm uses the compass directions as axes that define the domain of the search space. Each axis in the search space represents a variable in the objective function. Compass search moves in directions parallel to the axes in search for variable values that result in a better objective function value. One iteration of compass search is the evaluation of the compass directions plus choosing whether to step to another point or reduce the step length.

The compass search algorithm is best explained when we have a two dimensional search space, but can of course be applied to larger problems. In two dimensions the compass search starts in a base point with two variables as coordinates. An evaluation of this base point is completed to get a current best objective function value. An initial step length is then used to alter each of the variables in positive and negative direction. For the two dimensional case this result in four new points in the search space. We can imagine using the compass directions east, west, north and south as search pattern. These four new points are evaluated and their objective function values are compared to the objective function value in the base point. If the algorithm finds a better objective function value in one of the directions, the algorithm moves to this new point and then starts a new iteration. If no better objective function value is found by searching in the compass directions, the step length is halved and a new iteration starts. The algorithm continues with this compass search around the current best point until the step length is reduced to a value less than a given tolerance. This is the termination condition and the algorithm is finished. The algorithm result in an approximate solution of a best objective function value. Other termination conditions are also possible: e.g.

a maximum number of simulations can be set to ensure the algorithm does not continue indefinitely.

2.4.1 Compass search evaluation approaches

We say that compass search evaluates in directions parallel to the axes of the search space, and if it finds a better objective value it moves to this new point. The bullet points below show that there are different approaches to what is required before the algorithm move to a better point [6]. The details here are important for the implementation of the algorithm.

- Evaluate *all* the directions in order and move in the direction that gives a better function value.
- Evaluate *all* directions in parallel, if the implementation allows it. When all evaluations are finished, move in the direction that gives a better function value.
- Evaluate the directions in order and move in the direction that *first* gives a better function value.

Fig. 2.1 shows a flowchart of compass search for the first approach.

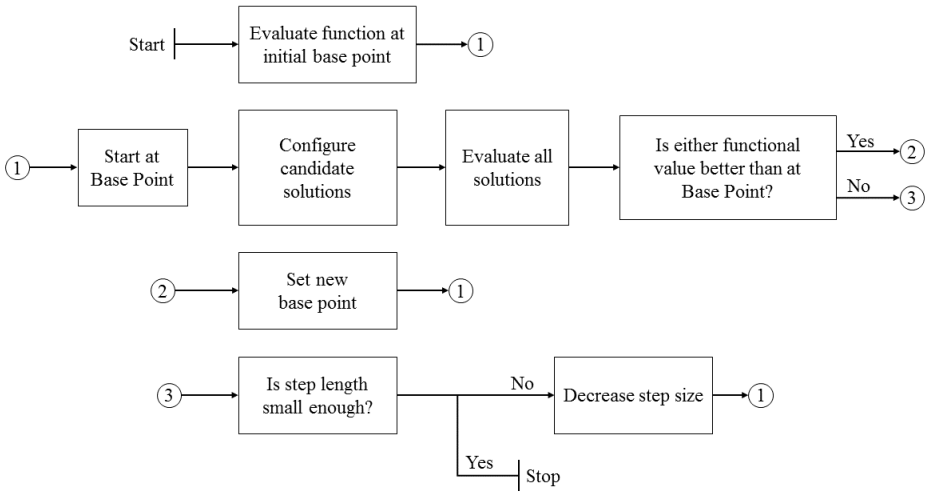

Figure 2.1: Flow chart of Compass Search.

2.5 Hooke-Jeeves Direct Search

In their paper [4], Hooke and Jeeves have the following definition of direct search:

Algorithm 1 Compass Search Algorithm for maximizing f. Taken from [7].

procedure CompassSearch(f, $\mathbf{x}_0$, Δ_{tol}, Δ_0, $\mathcal{D}$)
  $f_{best} \leftarrow f(\mathbf{x_0})$ ▷ Set initial value for f_{best}.
  $\mathbf{x} \leftarrow \mathbf{x}_0$ ▷ Set initial value for $\mathbf{x}$.
  $\Delta \leftarrow \Delta_0$ ▷ Set initial step length.
  while $\Delta > \Delta_{tol}$ **do** ▷ Iterate while step length is greater than tolerance.
    $\mathbf{M} \leftarrow (\mathbf{x} + \Delta \times d_k)$ **for all** $d_k \in \mathcal{D}$ ▷ Create list $\mathbf{M}$ of moves $\mathbf{x}_k$.
    $\mathbf{x}_{max} \leftarrow \text{argmax } f(\mathbf{x}_k)$ ▷ Find best move in iteration.
    if $f(\mathbf{x}_{max}) > f_{best}$ **then** ▷ Found a better position.
      $\mathbf{x} \leftarrow \mathbf{x}_{max}$ ▷ Change the "best" position.
      $f_{best} = f(\mathbf{x}_{max})$ ▷ Change the "best" objective value.
    else ▷ Did not find a better position.
      $\Delta \leftarrow 1/2 \times \Delta$ ▷ Reduce the step-length.
    end if
  end while
end procedure

> " We use the phrase 'direct search' to describe sequential examination of trial solutions involving comparison of each trial solution with the 'best' obtained up to that time together with a strategy for determining (as a function of earlier results) what the next trial solution will be. "

This quote has been repeated many times in literature regarding pattern search/ direct search, and as previously mentioned, these names are swapped a lot. In this section we will look into Hooke and Jeeves' definition of direct search. The explanation of the algorithm is mainly retrieved from their paper. Some other sources will also be used as reference because they provide interpretations of the algorithm which makes them easier to explain and implement.

Hooke-Jeeves direct search combines two type of operations. The operations can be considered as strategies, and together the two strategies bring the algorithm forward. By following the strategies, the Hooke-Jeeves direct search moves from base point to base point. The two strategies are named *exploratory moves* and *pattern moves* [4].

An initial base point, b_1, is chosen. This is where the algorithm starts from. The coordinates of the point are made up of the function's variables. Like in compass search, an evaluation of the base point is completed. The evaluation result in an objective function value for this point. From this base point the algorithm conducts *exploratory moves* to analyze the search space around the base point. This is done in almost the same way as an iteration of the compass search. If a better objective function value is found during the exploratory moves, the algorithm conducts a *pattern move* in the successful direction. A flow chart of the Hooke-Jeeves direct search can be found in Fig. 2.2.

Exploratory moves alters the variables of b_1, one at the time, in search for a better objective function value. The variables are altered by a given step length.

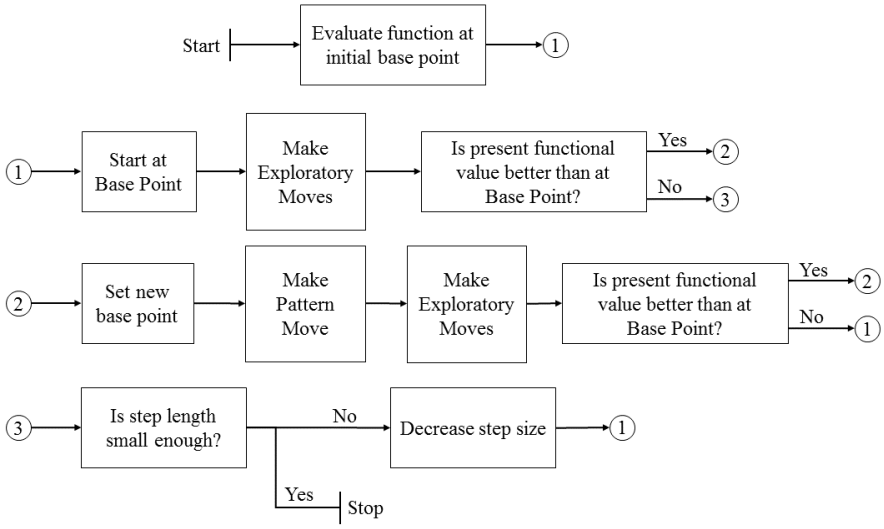

Figure 2.2: Flow chart of Hooke-Jeeves Direct Search.

For each variable, the algorithm evaluates the objective function when the variable is altered with the step length in the positive direction. If this does not give a better function value, the negative direction is evaluated. After the positive, and possibly the negative direction is evaluated, the algorithm continues to the next variable. This means that if a better function value is found in the positive direction, the negative direction will not be considered.

The exploratory moves are conducted to investigate the search space of the objective function within the perimeter defined by the step length. Hooke and Jeeves define the exploratory moves as simple, because in each move only one of the variables in the objective function change. The objective function value is evaluated based on this simple move, and then the algorithm continues to the next variable.

Pattern move is an operation done to investigate further in the direction that was found promising during the exploratory moves. If the sequence of exploratory moves are successful there will be a new point in the search space, b_2, with a better objective function value compared to the base point b_1. The pattern move uses the knowledge of a better objective function value in b_2 and wants to explore the search space in this assumed better direction. The distance and the direction from the base point b_1, to the new point b_2, define what is used for conducting the pattern move. A trial point is created two times the distance from the base point, in the direction found through exploratory moves [8]: $b_t = 2b_2 - b_1$. The objective function value of b_t is not evaluated. Instead, the algorithm starts doing exploratory moves around b_t. When the exploratory moves around the trial point is completed, its objective function value is compared to the objective function value

in b_2. If this objective function value is better than for b_2 the pattern move was successful and the point found through exploratory moves around b_t is set to b_3. The algorithm continues with creating the trial point $b_t = 2b_3 - b_2$ and conduct exploratory moves for b_t. If the first pattern move was not successful, b_2 is set as base point and we start a new round of exploratory moves.

Hooke and Jeeves write in their paper that they use "a strategy for determining (as a function of earlier results) what the next trial solution will be" [4]. With this they propose that they use the knowledge they acquire through exploratory moves as basis for the next step in the search. The pattern move is based on the assumption that because there was a better function value in this direction, we want to investigate more in this direction of the search space.

Step length reduction is done when exploratory moves around a base point (not a trial point) does not find a better objective function value. The step length is halved and exploratory moves start again for the current base point.

Termination of the algorithm is the same as for compass search. The algorithm terminates when the step length falls below a given tolerance or the algorithm reaches a maximum number of evaluations. The current base point becomes the best case result.

Algorithm 2 Hooke-Jeeves Direct Search Algorithm for maximizing f.

procedure HOOKE-JEEVESDIRECTSEARCH(f, $\mathbf{x_0}$, Δ_{tol}, Δ_0)
    $f_{best} \leftarrow f(\mathbf{x_0})$ ▷ Set the initial value for f_{best}.
    $f_{current} \leftarrow f(\mathbf{x_0})$ ▷ Set the initial value for $f_{current}$.
    $\mathbf{x} \leftarrow \mathbf{x}_0$ ▷ Set initial value for $\mathbf{x}$.
    $\Delta \leftarrow \Delta_0$ ▷ Set initial step length.
    while $\Delta > \Delta_{tol}$ **do** ▷ Iterate while step length is greater than tolerance.
        $f_{current}$ = EXPLORATORYMOVES($f_{current}$) ▷ Do exploratory moves.
        if $f_{current} > f_{best}$ **then** ▷ Found a better position.
            f_{best} = PATTERNMOVES($f_{current}, f_{best}$) ▷ Do pattern move.
        else ▷ Did not find better position.
            $\Delta = 1/2 \times \Delta$ ▷ Reduce step length.
        end if
    end while
end procedure

function EXPLORATORYMOVES(f_{input})
    $f_{output} = f_{input}$ ▷ Set input position as temporary output position.
    for all x_i in $\mathbf{x}$ **do** ▷ Go through all x_i in $\mathbf{x}$.
        if $f_{input}(x_i \pm \Delta) > f_{output}$ **then** ▷ Found better value when altering x_i.
            $f_{output} = f_{input}(x_i + \Delta)$ ▷ Change the output position.
        end if
    end for
    return f_{output} ▷ Return the new position.
end function

function PATTERNMOVES($f_{current}, f_{best}$)
    while $f_{current} >= f_{best}$ **do** ▷ Continue doing pattern moves.
        $f_{trial} = 2 \times f_{current} - f_{best}$ ▷ Create trial point.
        $f_{result=}$ EXPLORATORYMOVES(f_{trial}) ▷ Do exploratory moves.
        if $f_{result} > f_{current}$ **then** ▷ If exploratory moves found better point.
            $f_{best} = f_{current}$ ▷ Change base point.
            $f_{current} = f_{result}$ ▷ Change current best point.
        end if
    end while
    return f_{best} ▷ Return the current best point.
end function

Chapter 3

Petroleum production optimization using perforations

This chapter presents the optimization problem we are trying to solve in this thesis. We want to optimize the placement of perforations in a well for a given objective function value. In this chapter are explanations of how we want use optimization methods for this problem. We also discuss why we have chosen pattern search methods for this type of optimization.

3.1 Simulation-based optimization

A lot of engineering work has become simulation-based, and consequently there are optimization problems that do not have information regarding the derivative of the objective function. There are also cases where the derivative of objective function cannot be calculated. For these cases, derivative-free methods are very useful. When doing simulation-based optimization, a simulation must be run for every time the algorithm require more data to do a next optimization operation. [6]

Optimization of petroleum production is a case of simulation-based optimization. Studying future petroleum production, the placement of wells, or the settings of well completions all depend on the analysis of simulation results. A lot of reservoir engineering work is to create, run and analyze reservoir simulations.

3.2 Problem formulation

3.2.1 Objective of the optimization

When running reservoir simulations, we can require certain measurements in the result data. These measurements are based on what part of a simulation case

we want to study. In petroleum production optimization we use one or more of the simulation results as the objective function. The most common objective functions to optimize for is, as mentioned in chapter 2, net present value, cumulative productions, or combinations of the different measures. In this thesis, pattern search algorithms will suggest different candidate perforation configurations. Each location will be evaluated by running a simulation with this given perforation configuration. For each simulation we will get an objective function value that is to be compared to other evaluations. Based on the evaluations of the candidate configurations the algorithm search towards a best case objective function value and perforation configuration.

3.2.2 Optimizing well completions

Well completions are hardware that are placed in the borehole after drilling [9]. There are many different types of well completion equipment. Stand-alone, or in combination with others, well completions can contribute to a better exploitation of the reservoir. The well completion type under investigation in this thesis is perforations.

Perforations are openings in the cement and casing which result in a flow path from the near reservoir and into the wellbore [10]. This flow path is created after the well is completed with casing and cement. The perforations are created by using a perforation gun. Perforation guns send out a high-pressure unidirectional jet which include a cone of explosives [11]. The small explosion creates a cone shaped hole in the casing (see Fig. 3.1). It is important that the flow path in the perforation itself has a higher flow capacity than the flow from the formation [10]. The advantage of using perforated completions is that the location of inflow from formation into the well can be controlled. By analyzing log data, the perforations can be placed in well sections that does not contain e.g. water, gas or weak intervals. Reservoir properties can change along the well and perforating the whole well may not always be beneficial. Another advantage is that perforations can be sealed off if they are not producing correctly. Spending some time to optimize the locations and settings of well completions can give huge benefits for the well production in the long run.

3.2.3 Perforation handling

In this thesis we assume that a section of the well is split into a number of segments. Within each segment a perforation is to be placed to find a best objective function value. Placing a perforation interval differently within a segment can give different simulation results. The initial solution is that each perforation completely covers its respective segment (see Fig. 3.2). A complete coverage may not give the best objective function value. Therefore, the algorithm search for other perforation configurations. A possible solution may be like the one in Fig. 3.3.

The only thing limiting the placement of a perforation is if: 1. a perforation interval crosses the segment boundaries, or 2. the start and end of a perforation interval change order. Each perforation is given a segment of the well, and for a

Figure 3.1: A vertical well with cemented and perforated casing.
Taken from [11].

perforation to overlap another perforation it must first cross the segment boundary. The case of two perforations overlapping each other is handled in 1. Fig. 3.4 and 3.5 illustrate cases where the perforation configurations are not physically valid.

3.3 Pattern search methods in use

Well completion placement is a practical problem and therefore a good candidate for pattern search methods. Placing perforation configurations is not complicated. We simply need to find the correct position for the start and the end of the perforation-interval. As mentioned in chapter 2, the pattern search methods are easy to adapt to a wide range of problems. This is because they have a clear approach for iterating towards an approximate solution. The two algorithms applied in this theses are considered easy to implement because of the simple nature of their operations. M. Avriel argues that even if Hooke-Jeeves direct search requires many function evaluations, it is still straightforward to implement and a "reliable method" [12].

The pattern search methods find the best approximate solution. They are therefore useful as a first approach, and for finding an area of the search space where a global best objective function value exist [4]. Optimizing with these algorithms require that we have an initial step length and a minimum step length. The result of the optimization is dependent on the size we set for these two values. Let's consider an example where the minimum step length is relatively small compared to the initial step. A possible consequence of this dependency is that the algorithm will have to run many simulations before it terminates at a best objective function value. It may also be problematic to use a step length that is large compared to the segment lengths. A big step length can be the reason that the start and end of the perforation change order.

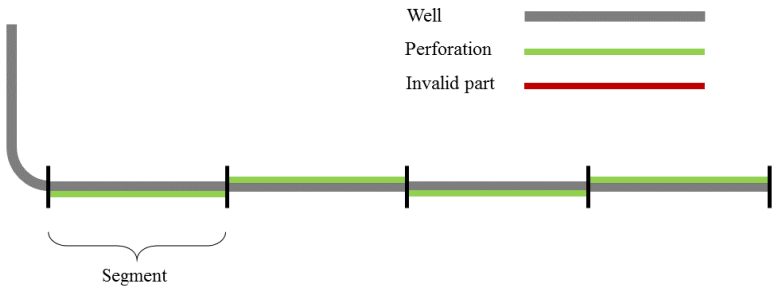

Figure 3.2: The base case with four segments and complete perforation coverage.

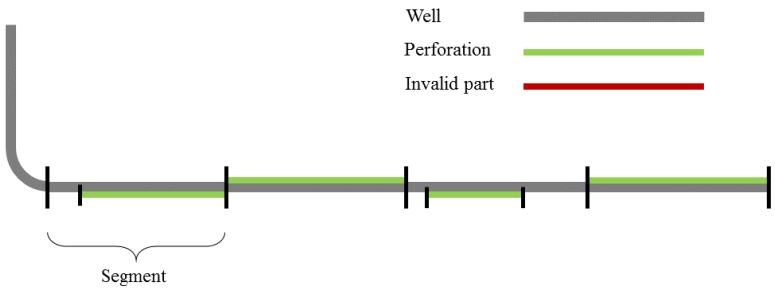

Figure 3.3: Valid perforation configuration.

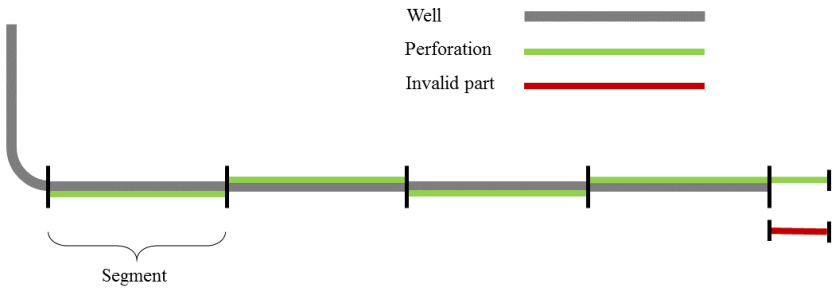

Figure 3.4: Invalid perforation configuration: Not physically feasible.

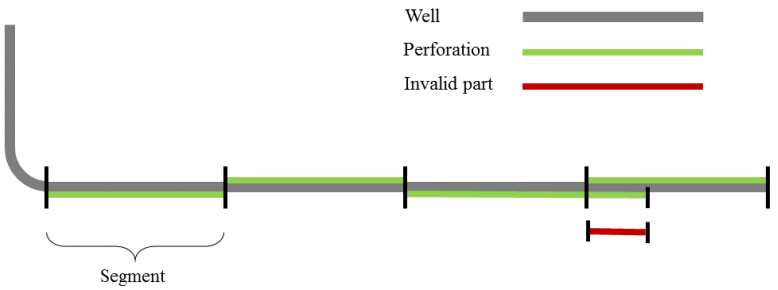

Figure 3.5: Invalid perforation configuration: Overlapping perforations.

Chapter 4

Implementation

This chapter concerns the implementation of Hooke-Jeeves direct search to an existing Ocean plug-in for Petrel. First is a presentation of the software and tools being used, followed by an introduction to object-oriented programming, which is the programming paradigm that is being utilized for Ocean plug-ins. The rest of the chapter describes the existing plug-in and how it was changed to add a second pattern search method.

4.1 Software

Petrel is an exploration and production software platform from Schlumberger [13]. The Petrel platform is able to run the ECLIPSE reservoir simulator [14], which is used to simulate the candidate perforation configurations generated by the optimization algorithms.

Ocean is a software development framework used to create plug-ins for the Petrel platform [15]. A software framework is "a reusable set of libraries or classes for a software system" [16]. With the Ocean framework we can create custom extensions to the Petrel platform, specialized for our solutions. We are given access to modify the Petrel objects and create our own processes. Ocean enables the developers to create new solutions and dedicated workflows for different exploration and production problems. The Ocean framework is built on top of the .NET development platform [15]. .NET is a software framework developed by Microsoft that enables developers to build applications for the Windows operating system [17].

C# is the programming language used by the Ocean framework. C# is an object-oriented language [18], the concepts of which is presented in the next section.

Microsoft Visual Studio is an integrated development environment (IDE) for creating applications. Visual Studio supports the C# programming language and is required for developing Ocean plug-ins. [19]

4.2 Object-oriented programming

> " Object-oriented design processes involve designing object classes and the relationships between these classes. These classes define the objects in the system and their interactions. When the design is realized as an executing program, the objects are created dynamically from these class definitions. " [20]

Object-oriented programming is a programming approach were the code is structured into classes [21]. A class can be seen as the "recipe" for defining the properties and functionality of an object. The properties of an object are often called *attributes* or *fields* in the class. In addition to its attributes, a class can include *methods*. Methods can be seen as predefined rules for changing the object's attributes and thus handles the behavior of the object [21]. A required method in a class is the *constructor*. The constructor has the same name as the class itself, and we use the constructor to create an instance of the defined object [22].

The Ocean framework provides a library of predefined classes. The library gives us access to and descriptions of the classes' attributes and methods, enabling us to manipulate Petrel data.

4.3 Existing plug-in

By using the software and techniques described above, we can create Ocean plug-ins for the Petrel platform. As mentioned in section 4.1, an Ocean plug-in can be seen as an extension of the tool collection in Petrel. We typically create plug-ins to solve specific problems that Petrel is otherwise not able to solve. For this thesis we want to use optimization algorithms that are currently not available in Petrel. The problem formulation for this thesis was defined on the basis of an existing Ocean plug-in created by PhD student Einar Baumann in the summer of 2015 at the Department of Petroleum Engineering and Applied Geophysics, NTNU. The existing Ocean plug-in optimizes the placement of perforations based on a given objective function using the compass search algorithm. Through this thesis the plug-in has been extended with a second pattern search algorithm: Hooke-Jeeves direct search.

4.3.1 Workstep

The plug-in functions as a workstep for the Petrel Workflow editor. The Workflow editor in Petrel is a tool where a collection of worksteps can be put in a sequence and executed. We can compare the sequence to a set of code lines created by a software developer. In the workflow editor, each code line is customized to do a predefined

operation. For instance, Petrel has a group of worksteps for well engineering tasks: e.g. *Well path design*, *Well completion design*, *Define well segmentation*. Each workstep performs an individual operation on a given input well. A workstep can have several input and output parameters.

With the Ocean framework we can create custom worksteps. We can specify our own input and output parameters, and create our own combination of operations. The workstep can contain functionality to change Petrel objects and run Petrel processes, or we can create custom objects and operations. The existing plug-in by E. Baumann is implemented as a custom workstep that optimize the locations of perforations in a well.

Input parameters For this optimization workstep, a set of input parameters are required. A Petrel *well* item with perforation(s) is needed as target for the optimization.

To run simulations of the candidate solutions suggested by the algorithm we need a Petrel *case* item. Having access to the case object we are able to run ECLIPSE simulations for each candidate configuration, and collect the results. Through the case object, we also have access to the strategies for the specific case. A *strategy* in Petrel is used to inform the simulator how the field changes over time. It can include information regarding the rates or pressures of production and injection wells, or other well changes that will impact a simulation run. [23]

We also need a *resolution* to set how many perforation segments the well is to be split into. The compass search algorithm needs the following parameters to begin and finish the optimization run: *initial step length*, *tolerance*, and *maximum number of simulations*. This results in a total of six input parameters for the workstep. Fig. 4.1 shows how the workstep looks like in the Workflow editor.

Figure 4.1: Workstep in Petrel Workflow editor.

In addition to these six input parameters, the workstep contains a form (see Fig. 4.2) used to specify the objective function expression for the optimization run. In this form we set whether to maximize or minimize the chosen objective function. The objective function can either be a simple value chosen from a collection of common simulation results, or an expression on the form

$$f = result_a \pm x \times result_b$$

4.3.2 Classes

Optimizer We can consider the `Optimizer` class as the engine for this optimization plug-in. The `Optimizer` holds all required information regarding the base case, the current best case and input arguments from the workstep. Based on the input

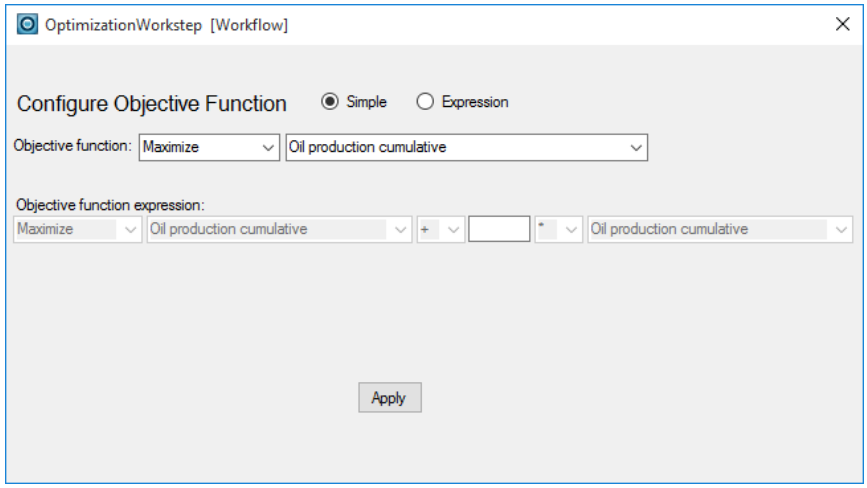

Figure 4.2: Form for optimization workstep.

arguments, the `Optimizer` creates a new instance of either the `CompassSearch` or the `HJDirectSearch`, and pass down the input arguments. The `Optimizer` also launches when a new perforation configuration can be applied to the well and is ready for simulation. It also handles the result after simulation. Because of the structure in the `Optimizer` class, only some small modifications were necessary to implement a second pattern search algorithm.

OptimizerCase The `OptimizerCase` encapsulates the Ocean class `Case` as an attribute. This `OptimizerCase` extends the functionality of a case to also include a `Perturbation`, a `Well` and a `Strategy`. When a perturbation is to be evaluated, the `Perturbation` field in the `OptimizerCase` is changed. Then the `Perturbation` is applied as perforations to the `Well`, and a simulation is run for the `Case`.

4.3.3 Handling of perforation configuration candidates

Through the Ocean framework we are able to access the perforationsfor a Petrel well. We can delete existing perforations and we can create new perforations. To create a new perforation Ocean needs three input parameters: a *name*, a *startMD*, and an *endMD*. MD is an abbreviation for *measured depth* [24]. This depth is used as a measure from the start to the end, along the length of the well. In contrast, we have the measure *true vertical depth*, TVD, which is different from MD when the well is not strictly vertical (see Fig. 4.3). MD is necessary when we need to specify locations in inclined wells. We assume in the rest of this thesis that positive direction is when the MD increases, while negative direction is when MD decreases.

The short list of required input parameters makes it relatively easy to modify perforations for a well. The challenge lies in handling the many different candidate perforation configurations that the algorithm wants to evaluate. A class named

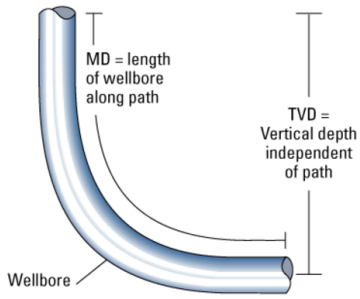

Figure 4.3: TVD vs. MD.
(Taken from [24].)

`Perturbation` is used to represent a candidate perforation configuration. A perturbation object have a list of startMD and endMD values: Variables = {startMD1, endMD1, startMD2, endMD2, ..., startMDn, endMDn}. In addition, the perturbation object holds information about the objective function and its value. In the code, a perturbation object named BestCasePerturbation holds the current best perforation configuration and its objective function value. The optimization algorithm generates perturbation objects with altered variable values, searching for a configuration that yield a better objective function values. This is handled by creating a copy of the BestCasePerturbation and then changing one of the variable in the list of MDs. A candidate perturbation is then ready to be applied as "real" perforations to the well.

4.3.4 Compass Search

Sequence diagrams of the Optimization plug-in when using compass search can be found in Fig. 4.4 and 4.5. See also Fig. 4.6 for a class diagram. During the initialization of the `Optimizer`, the well is split into segments based on the given resolution. Additionally, an `OptimizerCase` instance named BaseCase and an instance of `CompassSearch` are created. When the optimization starts, the `Optimizer` object requests a new `Perturbation` from the `CompassSearch` object. `CompassSearch` may have a perturbation object ready for evaluation in the NewPerturbations queue. If not, `CompassSearch` checks if the last iteration of evaluations found a better perturbation and either step in this direction or reduce the step length. New perturbations are then perturbed and put in the NewPerturbations queue. `CompassSearch` returns a perturbation from NewPerturbations which is ready to be applied to the well. Then a simulation with this perforation configuration starts.

When the simulation is completed (Fig. 4.5), the `CaseMonitor` object calls the `PostSimulationCS()` method in the `Optimizer`. The results are collected and the best case is updated if a better objective function value has been found. Then the `Optimizer` calls the `EvaluateNextCS()` method again, and this result in a

loop. The loop stops when the `steplength` attribute has a lower value than the `tolerance` attribute.

Creating perturbations In the original compass search, two perturbations are made for each perforation. The first perturbation alters the startMD in the positive direction, and the second perturbation alter the endMD in the negative direction. This implementation narrows the search towards the center of the segment. The startMD will for all candidate solutions have altered its value to a location closer to the center of the segment, and the same applies for the endMD. This results in two candidate solutions for every perforation interval. In the case of two segments we will have four MD elements in the variable list, which results in a total of four candidate perturbations. Hooke-Jeeves was implemented with a different approach to the creation of perturbations. For the two algorithms to be comparable, the compass search was modified.

Modification In the new version, compass search creates two perturbations for every single MD value in the list of variables. One in positive and one in negative direction. For the case with two segments the algorithm will now have a total of eight candidate perturbations. This new implementation will not have the center approach, but instead, as the algorithm progresses, each startMD and each endMD will move back and forth along the well.

Validity With the center approach it was not necessary to check if the candidate solutions were valid as perforations. We knew that the candidate perforation configurations shrinked towards the center at all time and it was not possible to move outside the segment. With the new approach we can get many candidate perforation configurations with MD values that are not physically feasible, as discussed in section 3.2.3. We therefore need to check the validity of all perturbations before they are applied as perforations to a well. If a perturbation is valid, it is queued for evaluation. Invalid perturbations are given the trivial objective value of -1000. With the validity check we are certain that only perforations that are feasible are used in the simulation.

We do not know if the center approach is better than the other, or if they give other results. The two different procedures just show how important it is to carefully define the implementation of the algorithm. Different handling of the variables may give different results.

Evaluation approach In chapter 2 we presented three different evaluation approaches for compass search. The implementation of compass search in this Ocean plug-in evaluates all compass directions, before moving to the point which have the best objective function value. Consequently, more simulations are required, but we are certain that the direction we move in is the best choice.

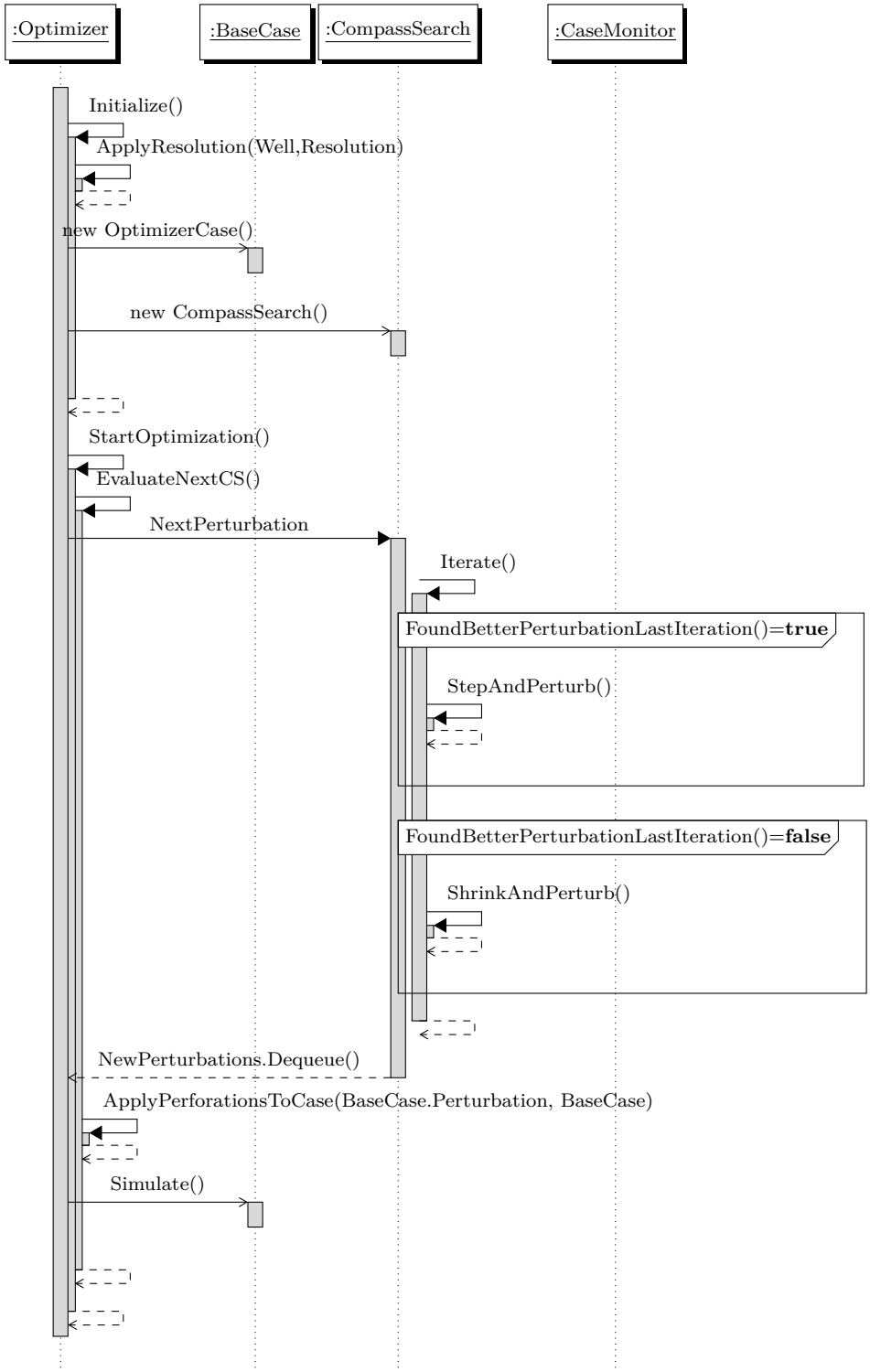

Figure 4.4: Sequence diagram for Compass Search part 1.

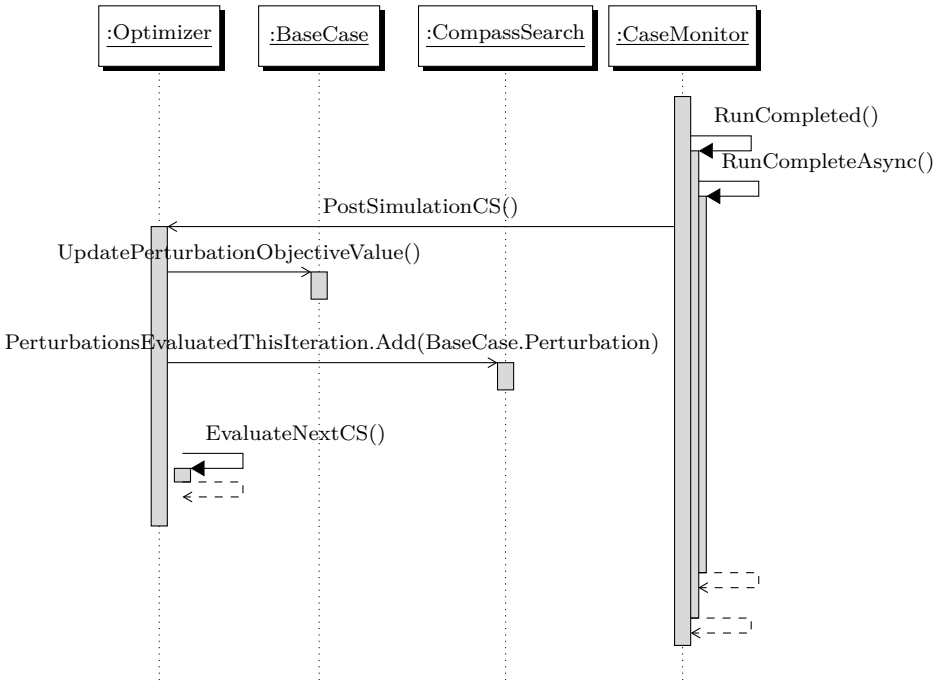

Figure 4.5: Sequence diagram for Compass Search part 2.

4.4 New plug-in

4.4.1 Inheritance

Inheritance is one of the main features of object-oriented programming. It is a method for structuring implementation of new classes based on existing classes. If used correctly, inheritance can contribute to clean and reusable code. [21]

When applying inheritance, we have a base class that includes attributes or methods which makes it easier to implement the derived class [22]. Other terms for the base class are *parent class* or *superclass*. The derived class is often referred to as the *subclass*. Inheritance result in less implementation effort because the defined attributes and methods in the superclass are available for the subclass. A derived class have the same functionality as its parent class, but can additionally have own attributes and methods.

We apply inheritance to this plug-in because we want to implement a second algorithm. Compass search and Hooke-Jeeves direct search have many properties and methods that are identical. Instead of duplicating code lines, a general `OptimizationAlgorithm` class was created. Both `CompassSearch` class and the new `HJDirectSearch` class inherits from this parent class.

4.4.2 Additional algorithm

Implementation of a second algorithm required a new class: `HJDirectSearch`. This class was implemented first, and copied the structure of the `CompassSearch` class. Since the two classes had many of the same attributes and methods, we decided to put their common attributes and methods into a parent class. A class diagram of this inheritance relationship can be found in Fig. 4.6.

OptimizationAlgorithm class The `OptimizationAlgorithm` class includes mainly the properties that are necessary for a pattern search algorithm. Both of the algorithms require an initial step length and a tolerance for termination. A method, `isFinished()` is used to check if the step length is below the tolerance.

HJDirectSearch class As mentioned in section 4.3.3, the compass search was changed because of the implementation of `HJDirectSearch`. For the exploratory moves, Hooke-Jeeves direct search will conduct a search for each MD value individually. It will first create a candidate perturbation in the positive direction, and if this perturbation is valid it will be evaluated. If the positive direction gives a better objective function value, the algorithm will go straight to the next MD variable in the list and discard the negative direction. There will only be created a perturbation for the negative direction if the perturbation in positive direction fails to find a better objective function value.

An integer `currentVariable`, in `HJDirectSearch`, controls which of the MD values in the `Variables` list that is to be altered and evaluated. This integer is increased as the algorithm processes the result for the positive, and maybe negative, direction. As long as the `currentVariable` is less than the total number

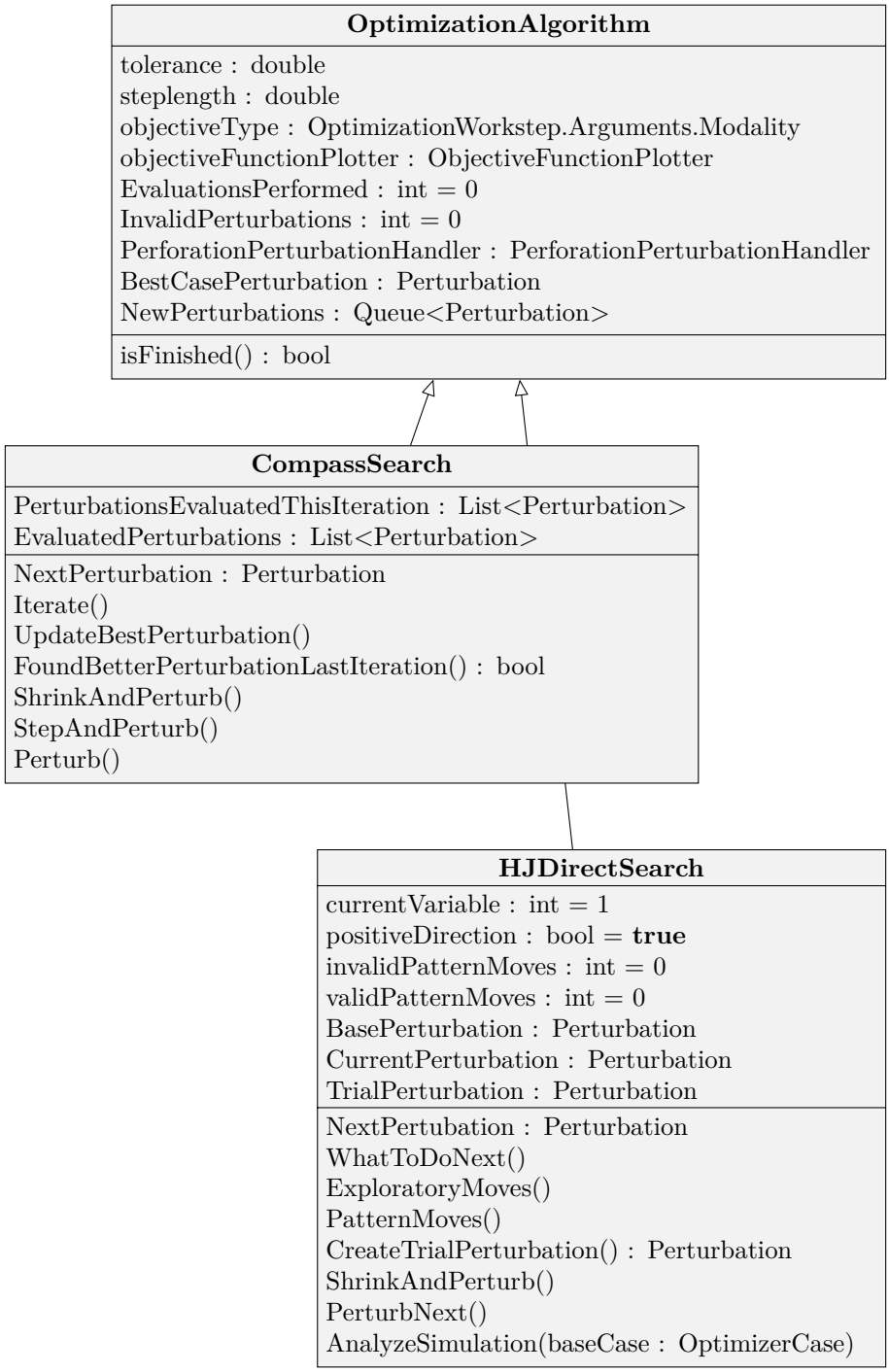

Figure 4.6: Class diagram showing the inheritance relationship.

of elements in the `Variables` list, `ExploratoryMoves()` are continued (see Alternative 1 in Fig. 4.9). When the `currentVariable` reaches the total number of elements in the list, the algorithm evaluates what to do next. The algorithm either does a `PatternMove()` or `ShrinkAndPerturb()` based on the results from `ExploratoryMoves()`. If it is decided to do a `PatternMove()`, a trial perturbation is created and exploratory moves for the trial perturbation is conducted. Otherwise, the `steplength` is halved and `ExploratoryMoves()` restarts with this new step length.

4.4.3 Algorithm tracking

To analyze the performance and efficiency of the algorithms, some different counters were implemented.

Candidate solutions Compass search was modified to alter the variables in both directions, and we therefore had to insert validation of the perturbations. Hooke-Jeeves direct search does not necessarily create a perturbation to check in the negative direction. To analyze the algorithms correctly, we found it appropriate to track the number of candidate perforation configurations suggested by the algorithms. In the code, this number is incremented for every time a new perturbation object is created.

Invalid perturbations After a perturbation is created we check the validity of it to make sure that if we apply it as perforations to a well it is physically feasible. We want to track the number of invalid perturbations to see if this can indicate anything concerning the work efficiency of the algorithm.

Simulations We track the number of simulations for the same reasons we track the number of invalid perturbations. In fact, the sum of invalid perturbations and the number of simulations are equal to the number of candidate solutions. Either a configuration is valid and will be simulated, or a configuration is invalid.

4.4.4 New choices in workstep form

As a consequence of the second algorithm, the user of the plug-in needs to be able to choose which algorithm to use. The algorithm choice is implemented as a drop down list in the workstep form. Fig. 4.10 shows the new look of the workstep form.

4.5 Optimize with two algorithms

The optimization plug-in can now run perforation optimization with two different algorithms. When using the two algorithms, compass search and Hooke-Jeeves direct search will start off in the same way. They will both explore the neighborhood of the current base point. When this search is completed the compass search will either move to a new point or reduce the step length. The Hooke-Jeeves direct

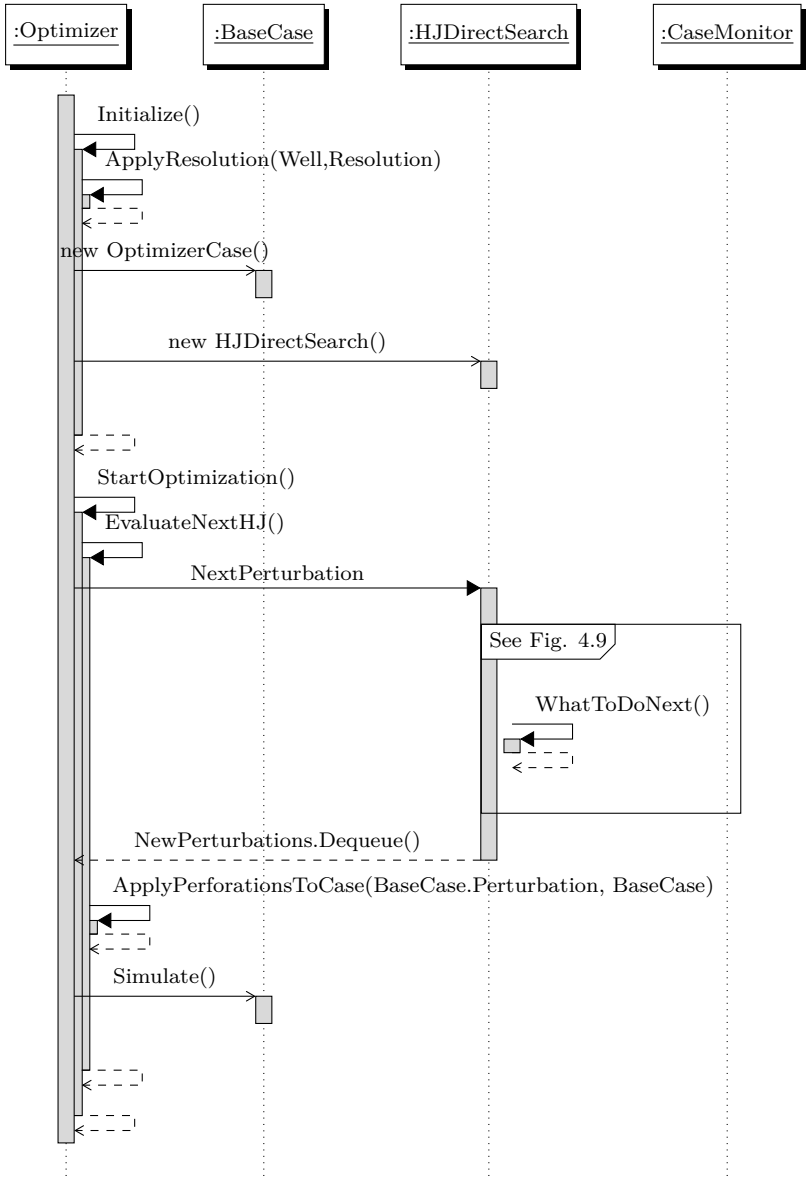

Figure 4.7: Sequence diagram for Hooke-Jeeves Direct Search part 1.

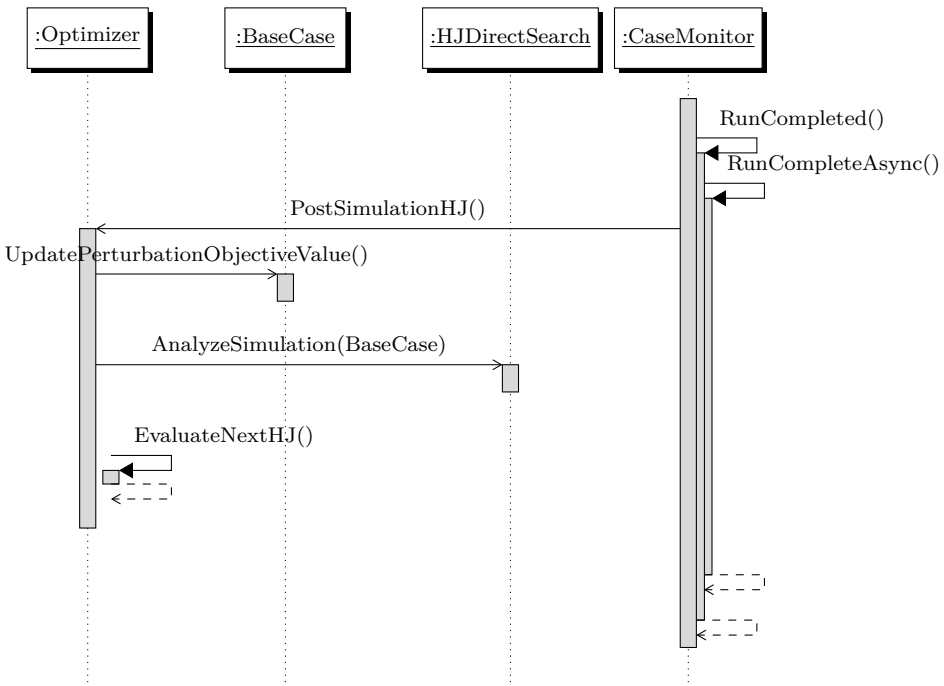

Figure 4.8: Sequence diagram for Hooke-Jeeves Direct Search part 2.

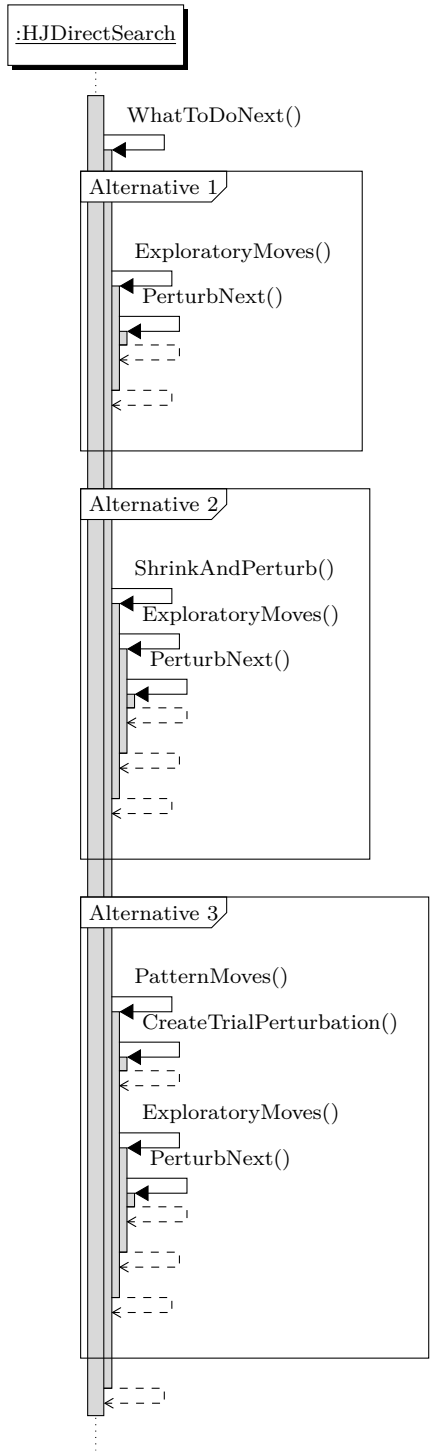

Figure 4.9: Sequence diagram for Hooke-Jeeves Direct Search part 3.

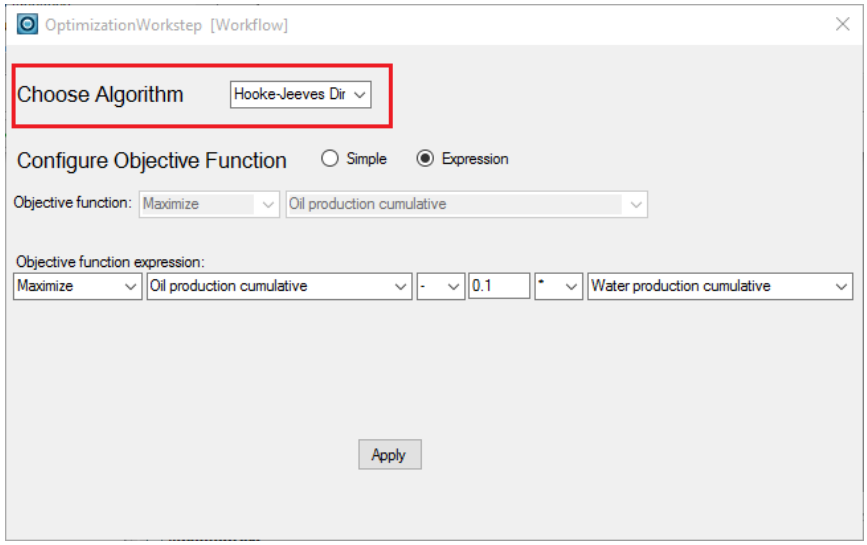

Figure 4.10: New form for optimization workstep.

search will do pattern moves if promising directions have been discovered, or it will reduce the step length. Compass search will take many small steps on the path of finding the best case, while Hooke-Jeeves have the opportunity to take long steps in the search space if such an optimum is detected.

In the following chapter we conduct two experiments by using the Optimization plug-in. We are curious to see the behavior of the algorithms when they are being applied to a simulation case, and we want to see if we find similarities or differences between the two algorithms when they are executed using the same parameters.

Chapter 5

Optimization plug-in in use

In this chapter we use the Optimization plug-in to do experiments with a simulation case with one production well and one injection well. The first experiment is done using only the Hooke-Jeeves direct search algorithm. In the second experiment we rerun some cases from the first experiment, using the compass search algorithm.

5.1 Optimization set up

5.1.1 Base case

The base case for the experiments is relatively uncomplicated. It includes one horizontal production well and one horizontal injection well. The production well is placed right above the injection well (see Fig. 5.1). Included in the base case is a strategy for determining the specifications for the wells. The injection well is controlled by a bore hole pressure (BHP) target of 3700 psia, while the production well is controlled by the liquid volume target rate of 3000 stb/day and has a BHP lower limit of 1500 psia. Initially, the production well is cased and has one perforation interval that spans from the heel of the well to the toe of the well. When applying the Optimization plug-in to the production well, this only perforation interval will be split into the number of segments that is given as resolution.

The base case used is a simple simulation case, but it is good enough for the testing and analysis we want to do in this thesis. If we were to use a more complicated case, the run time for each simulation would be longer.

5.1.2 Objective function

In the following experiments we want to maximize this objective function:

f = cumulative oil production - 0.1×cumulative water production

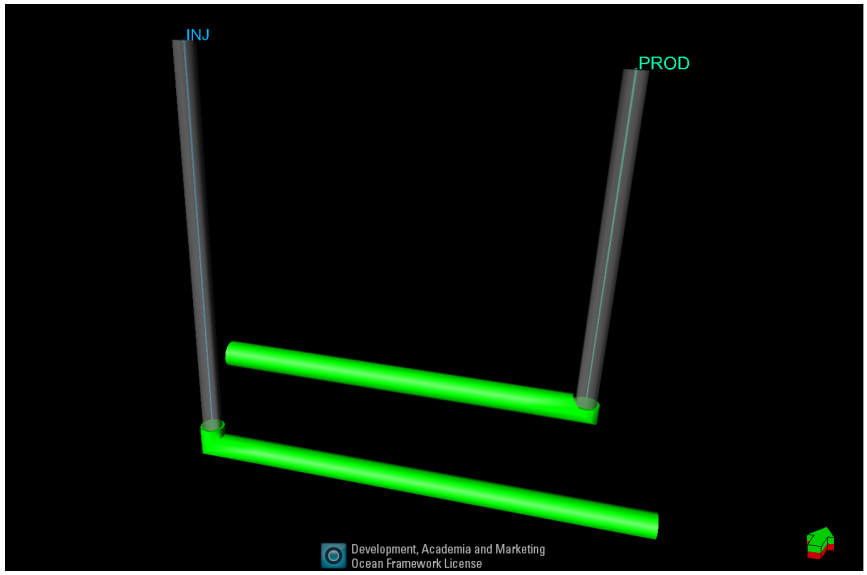

Figure 5.1: 3D view snapshot of base case well configuration.

5.1.3 Control variables

The control variables for this optimization is the list of MD values that represents a set of perforation configurations. For each perforation there are two control variables:

$$\text{Variables} = \{\text{startMD1, endMD1, startMD2, endMD2, ..., startMDn, endMDn}\}$$

$$\text{n} = \text{number of segments}$$

5.1.4 Constraints

The constraints for the control variables in these experiments is the start and end of the segments, and depends on the given resolution. For a detailed overview of the constraint bounds, see Table A.1 in Appendix B. This table shows the base case, with full perforation coverage, for different segment resolutions. A perforation interval is required to be located within its given segment bounds to comply with the constraints.

5.1.5 Input parameters

The required input parameters are the same for compass search and Hooke-Jeeves direct search.

Resolution/segments We use a different number of segments to study which resolution will be ideal for the given base case. A greater number of segments result in more control variables, hence more simulations, but may give a more precise perforation configuration. We want to see which segment resolution that results in the best objective function value, and to study how big is the effect of increased control variables on the total number of simulations.

All the optimization runs in these two experiments start with one perforation interval spanning from the heel to the toe of the production well. This single perforation is split into a given number of segments based on the input resolution. For the cases in these experiments we have decided to use 2, 4, 8 and 16 segments. This result in respectively 4, 8, 16 and 32 control variables. Fig. 5.2 shows well section windows of the production well with the different resolutions and full perforation coverage.

Tolerance The tolerance determines how detailed the last iteration of the search is. This is related to the precision of the best case perforation configuration when the optimization run is completed.

Initial step length The initial step length decides where to start the search in the search space. We want to study this parameter to see how different initial step lengths affect the objective function value and perforation configurations.

Maximum simulations This parameter is used as a termination condition of the algorithm. The experiments completed through this thesis have used a value for maximum simulations that was relatively high. This is because we wanted the algorithms to reach the tolerance limit and not be stopped by the maximum number of simulation.

5.1.6 Simulated evaluations ratio

Both algorithm implementations keep track of the number of candidate evaluations suggested by the algorithm, and how many of them are actually simulated. For each of the cases we will calculate the simulated evaluations ratio. We want to use this ratio to compare the two algorithms to each other.

$$Ratio = \frac{simulations}{candidates}$$

5.2 Experiments

The first experiment uses the Hooke-Jeeves direct search to find input parameters that gives the best objective function for the simulation case. In the second experiment we run some of the cases from experiment again, but changing to the compass search algorithm.

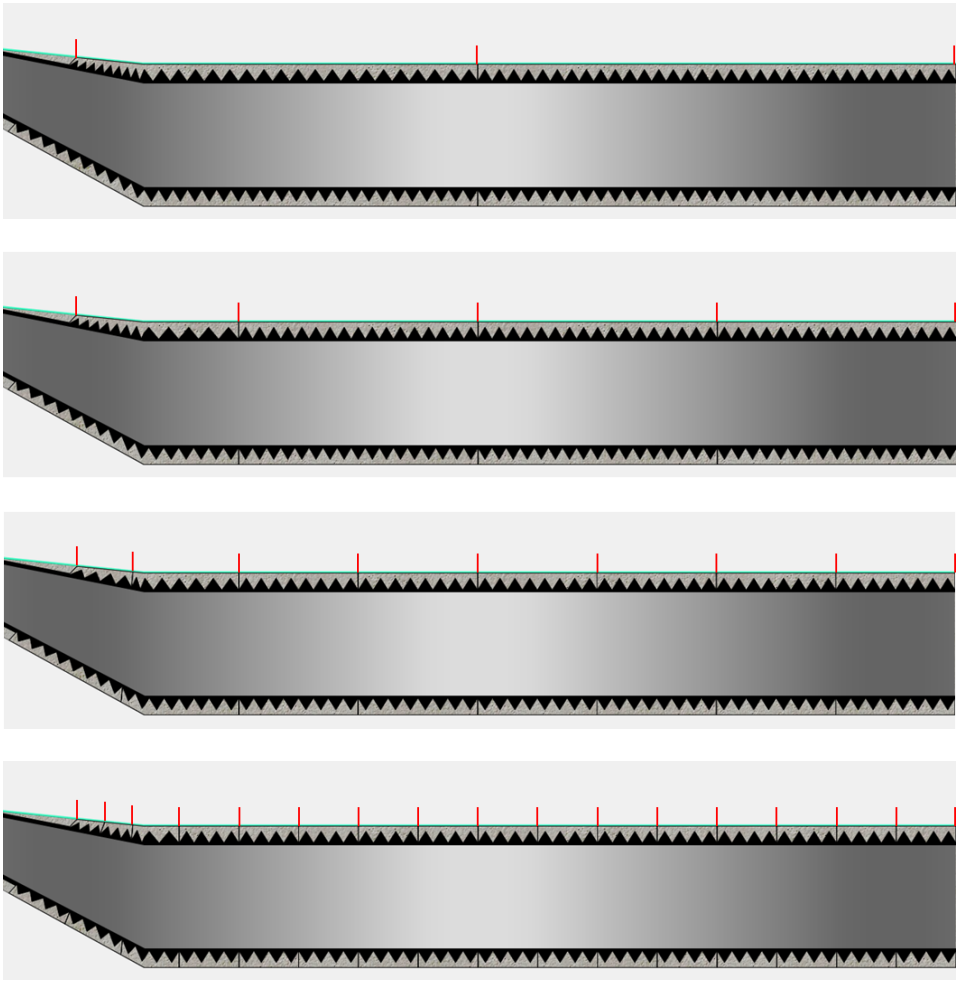

Figure 5.2: Well section image of different base case segment resolutions with full perforation coverage.

The parameters chosen for these experiments were found after running some test cases. We therefore know the actual segment lengths for the different resolutions. Based on this we choose the different initial step lengths and tolerance values.

5.2.1 Experiment 1

In this first experiment we use the Hooke-Jeeves direct search algorithm to study how change of the different input parameters affect the behavior of the algorithm.

Cases 1-4 In these cases, we want to study the effect of reducing the termination term, tolerance. There are four cases with two segments, and all the cases start with the same initial step length. We use four different tolerance values: {10, 5, 1, 0.5}. The assumption is that as the tolerance decreases, the more simulations will be run. We are also interested to study if the different cases result in the same best case perforation configuration and objective function value.

Cases 5-8 With cases 5-8 we want to study the effect of changing the initial step length. All four cases split the perforation into four segments, and we keep the tolerance fixed at 1. The different initial step lengths are: {45, 30, 25, 10}. For these cases we want to see if different initial step lengths result in the same best case, and discuss what can be the reason if they do not match.

Cases 9-11 In cases 9-11 we want to study the effect of increasing the number of segments. The optimization is run for 2, 4 and 8 segments. We keep the other parameters fixed for all three cases: the initial step length is 45, and the tolerance is 1. With these cases we are curious to see which resolution reaches the best objective function value and if any of them stands out. If so, how much better is the objective function value with this resolution.

Cases 12-15 Here we do the same as for cases 9-11, but we want to increase the number of segments even more. Because of the shorter segment length when we have 16 segments, the initial step length needs to be reduced to 15 meters. We keep the tolerance at 1, and the initial step length fixed for all four cases.

5.2.2 Experiment 2

We rerun seven of the cases in the first experiment with compass search instead of Hooke-Jeeves direct search. The results from using compass search are to be compared with the results from cases 5-8 and 9-11 with Hooke-Jeeves direct search. These cases were chosen for a rerun because they gave some interesting results in experiment 1, and we want see if the effect of parameter change is the same for compass search.

With the cases in this experiment we want to compare the two pattern search methods. We are curious to see if they result in the same perforation configurations and objective function values. It is expected that the compass search will run more

simulations and have more configuration candidates, but we do not know how big this difference is. Hooke-Jeeves only evaluate the control variable in negative direction if the positive direction does not find a better objection function value. Compass Search evaluates both directions regardless.

Cases 16-19 We run four optimization runs with four segments each and with the tolerance fixed at 1. The initial step lengths are 45, 30, 25 and 10.

Cases 20-22 We run three optimization runs with 2, 4 and 8 segments. The initial step length and tolerance are fixed at respectively 45 and 1.

5.3 Results

At first, we want to emphasize that with a different simulation case, optimization runs with the same algorithms and input parameters will give different results.

Best case perforation configurations for all cases can be found in Table A.2 - A.4 in Appendix A. The most important observations are presented in the following text.

For some of the cases we plot the objective function value against the number of simulations. We use simulations instead of candidate evaluations because simulations better show how much effort/time was used to reach the best case objective function values. The data for these plots are collected from the output of the optimization run. For every time the algorithm finds a new better objective function value, this objective function value and the current total simulation number is collected. Simulation number 0 is the objective function value from the base case evaluation. The last point in each measurement is the best case objective function value with the total number of simulations completed.

5.3.1 Experiment 1 results

Results regarding simulations and candidate evaluations for experiment 1 is presented in Table 5.1.

Case 1-4 comments With these four cases we expected that as the tolerance decreases, the more simulations are needed for the step length to reach the tolerance. The four optimization runs start with the same initial step length, 45, and only the "finish line" is different between the four cases. Our expectations are met and both the number of simulation and candidate evaluations increase as the tolerance decreases. We take a closer look at case 1 and 4 to find what differs between them. Case 1 needs 32 simulations to reach the best case objective value of 85840.77 $[m^3]$, while case 4 reaches 85866.96 with 61 simulations. This is only a difference of 26.19 $[m^3]$. By reducing the tolerance to a smaller value we see that there are possibilities for reaching a better objective function value, but in this case it does not really give a big effect. Even so, case 4 has the best increased objective function value of (85866.96 - 83908.11 =) 1958.85 $[m^3]$.

Case no.	No. of segments	Initial step length [m]	Tolerance [m]	Simulations	Candidates	Ratio [%]	Best case objective function value
1	2	45	10	32	53	60	85840.77
2	2	45	5	37	61	61	85840.77
3	2	45	1	52	85	61	85865.46
4	2	45	0.5	61	100	61	85866.96
5	4	45	1	248	306	81	90823.04
6	4	30	1	276	334	83	90818.74
7	4	25	1	216	279	77	90817.97
8	4	10	1	373	462	81	90822.66
9	2	45	1	52	85	61	85865.46
10	4	45	1	248	306	81	90823.04
11	8	45	1	291	487	60	90808.96
12	2	15	1	34	62	55	83926.98
13	4	15	1	271	356	76	90819.70
14	8	15	1	365	478	76	90820.00
15	16	15	1	613	1029	60	90819.39

Table 5.1: Experiment 1 results - Overview of cases with Hooke-Jeeves Direct Search.

If we study the best case perforation configurations for the four cases we see that case 1 and 2 reach the exact same solution. The reduction in tolerance from 10 to 5 does not actually impact the resulting best case configuration. The best case perforation configuration differs with approximately 2 meters for the startMD in the second segment when we compare case 1-2 to case 3-4. See table A.2 for details.

Case 5-8 comments For cases 5-8 we wanted to study the effect of changing the initial step length. One might think that with initial step length 45 it would require more simulations to reach the tolerance than it would with initial step length 10. This did not happen for these cases. It is actually the case with the lowest initial step length, 10, that uses the most simulations.

Cases 6 and 7 reach almost the same objective function values, but a closer look at the results show that they result in different best case perforation configurations (see Fig. 5.3 and 5.4). The best case perforation configurations in segment 1, 2 and 4 can be considered in the same neighborhood, but the perforation interval in segment 3 is very different between the two cases. These observations are the same for cases 5 and 8.

The observations of cases 5-8 show that by having different initial step lengths, the algorithm was unable to find the same optimum in the search space. One that finds the best optimum among all the experiments is case 5 which reached the highest objective value out of all the cases with 90823.04 $[m^3]$.

Figure 5.3: Best case perforation configuration case 6.

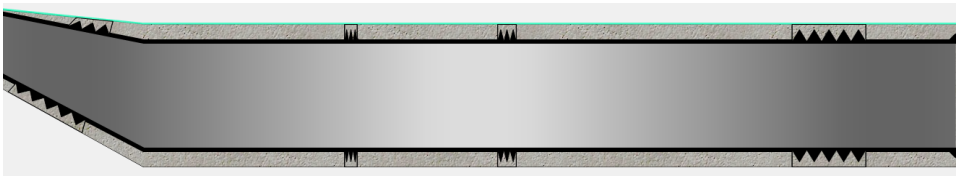

Figure 5.4: Best case perforation configuration case 7.

Cases 9-11 comments For cases 9 and 10 we reuse the results from case 3 and 5. Three cases are run with the same initial step length and tolerance, but with a different number of segments. Case 10, with 4 segments, reaches the best objective function value. The second best objective function value was found with 8 segments, which puts the case with 2 segments as the worst among these three cases.

If we look at total number of simulations, case 11 has the most. This is natural consequence of having more control variables with 291. Doubling the number of segments also doubles the number of control variables. We therefore consider the increased simulation numbers from case 9 to 11 as expected.

Cases 12-15 comments Initial step length 15 is not big enough for case 12 with 2 segment to find a good objective function value compared to the other three cases. Cases 13-15 find objective function values that are approximately the same.

We see the same effect regarding total number of simulations as in cases 9-11. More segment gives more control variables and consequently more simulations.

We want to compare the cases visually when it comes to best case perforation configurations (see Fig. 5.5 - 5.7). It looks like there is a trend for having more perforation coverage at the heel and at toe of the well.

Experiment 1 overall comments Based on the results from cases 1-15 it seems like the best resolution for this base case is four segments. More cases with eight segments and other input parameters may be conducted to investigate this resolution further.

None of the case runs with 2 segment reach an objective function value above 90000. Having 2 segments gives only four control variables, and maybe this resolution restricts the ability to specify enough in detail for the perforation configuration.

We have calculated the ratio of simulations divided by candidates for all cases in experiment 1. The ratios range from 55 to 81 % for cases 1-15. Cases 1-4 have

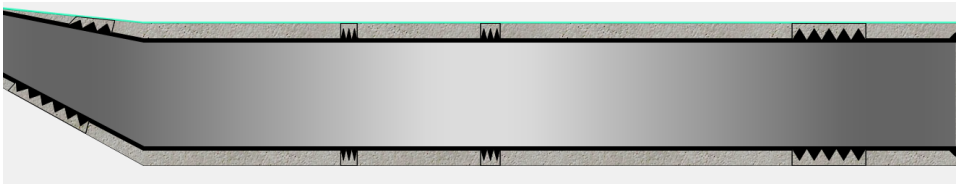

Figure 5.5: Best case perforation configuration case 13.

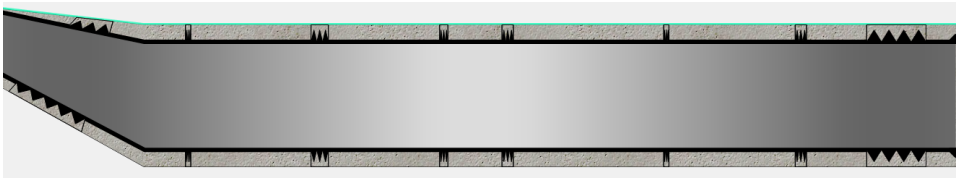

Figure 5.6: Best case perforation configuration case 14.

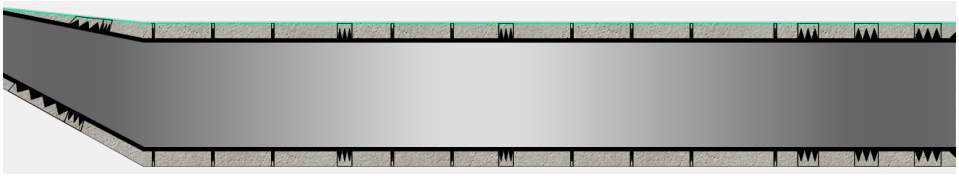

Figure 5.7: Best case perforation configuration case 15.

approximately the same ratio for all cases. The other groups of cases have varying ratios.

We found that the results in cases 5-8 and 9-11 gave the most interesting results and wanted to run the same cases with compass search.

5.3.2 Experiment 2 results

Results regarding simulations and candidate evaluations for experiment 2 is presented in Table 5.2.

Cases 16-19 comments When it comes to objective function, the two algorithms reach roughly the same values. For most of the case comparisons, the perforation configurations are very different, but one of the comparisons stand out. Case 8 with Hooke-Jeeves direct search and case 19 with compass search reach exactly the same objective function value, but have different perforation configuration in the second segment (see Fig. 5.8 - 5.9). The difference is not very big, but we see in the well section windows that they are not placed in the same location. A reason for the exact same objective function value may be that the two perforation configurations are placed within the same grid block. When the simulation is run, this will result in the same objective function value.

We have plotted cases 5-8 with Hooke-Jeeves direct search in Fig. 5.10, and cases 16-19 with compass search in Fig. 5.11. They were plotted in two separate

Case no.	No. of segments	Initial step length [m]	Tolerance [m]	Simulations	Candidates	Ratio [%]	Best case objective function value
16	4	45	1	267	384	70	90809.89
17	4	30	1	344	496	69	90820.34
18	4	25	1	358	512	70	90821.37
19	4	20	1	711	944	75	90822.66
20	2	45	1	54	88	61	85865.46
21	4	45	1	267	384	70	90809.89
22	8	45	1	579	1024	57	90808.66

Table 5.2: Experiment 2 results - Overview of cases with Compass Search.

Figure 5.8: Best case perforation configuration case 8.

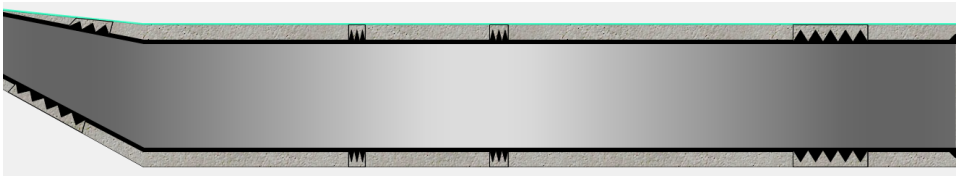

Figure 5.9: Best case perforation configuration case 19.

figures so that the curves do not overlap each other. The first we notice is that the curves for Hooke-Jeeves direct search cases take "large vertical jumps" for the objective function values before they even out in horizontal direction. This is in contrast to the compass search curves which have more "even edges". We know from the implementation that the compass search method moves to a neighboring point if this point have a better objective function value than the base point. Hooke-Jeeves on the other hand can take "long" pattern moves in a specified direction if such pattern moves continue to find better objective function values. This is probably the reason for the jagged curves for cases with Hooke-Jeeves direct search.

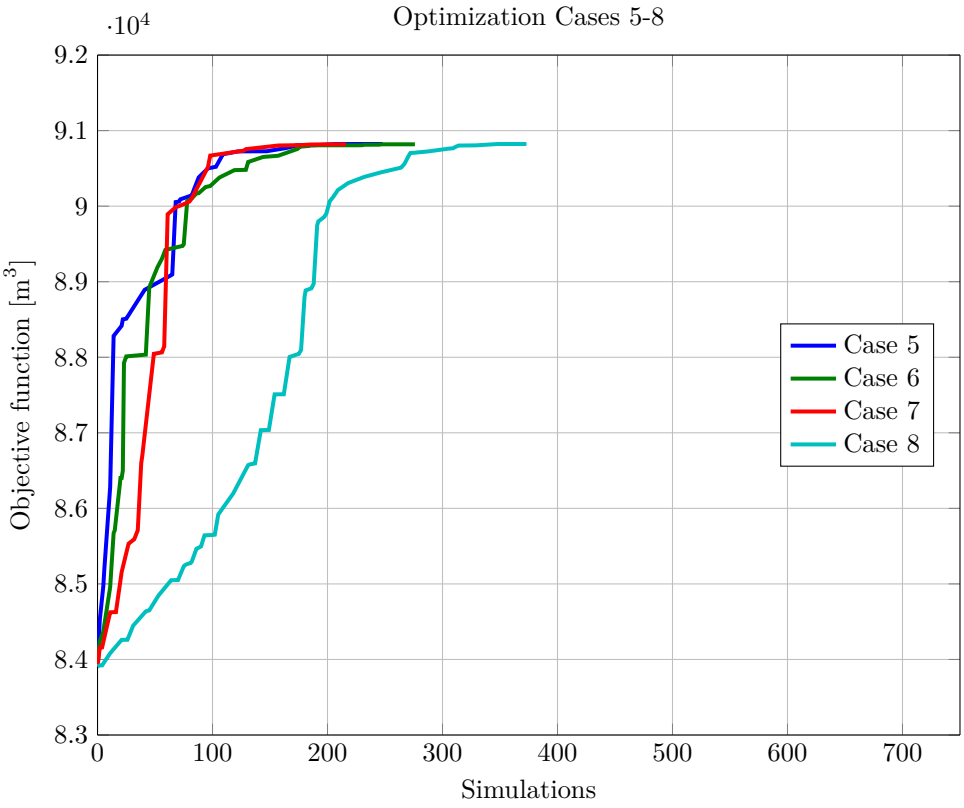

Figure 5.10: Plot of cases 5-8 with Hooke-Jeeves Direct Search.

Cases 20-22 comments The two algorithms reach the same objective function values for 2 segments (case 9 and 20), and approximately the same value for 8 segments (case 11 and 22). For 4 segments (case 10 and 21), the two algorithms have different best case objective function values.

When it comes to the perforation configuration Hooke-Jeeves direct search and compass search reach the same solution with 2 segments. In the cases with 4 and

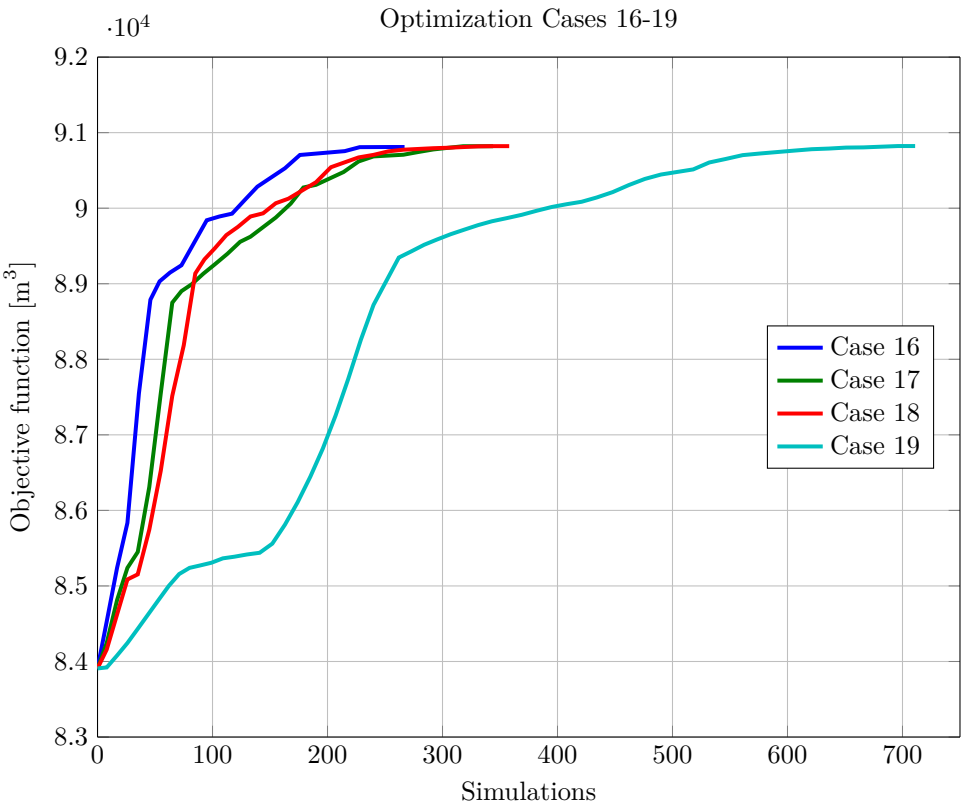

Figure 5.11: Plot of cases 16-19 with Compass Search.

8 segments, the two algorithms only have the same perforation configuration for their respective first segment. For the other segments, the algorithms have different perforation configurations.

The algorithms use approximately the same amount of simulations for 2 and 4 segments (52 vs. 54, and 248 vs. 267), but almost doubling the simulations for the case with 8 segments (291 vs. 579). Compass search produces twice as many candidate evaluations for the case with 8 segments.

Cases 9 to 11 and cases 20 to 22 are all plotted in Fig. 5.12. We see that the curve for case 20 approximately covers the curve for case 9, which is consistent with the identical best case objective function value and best case perforation configuration for the two cases. For cases 10 and 11 we again see the "vertical jumps" before evening out in horizontal direction for Hooke-Jeeves direct search. Cases 21 and 22 with compass search shows more rounded edges in their curves. The doubling of simulations when comparing case 11 and case 22 is very well illustrated in this plot. We see that the curve for case 22 does not reach the same approximate values for objective function until after case 11 is completed.

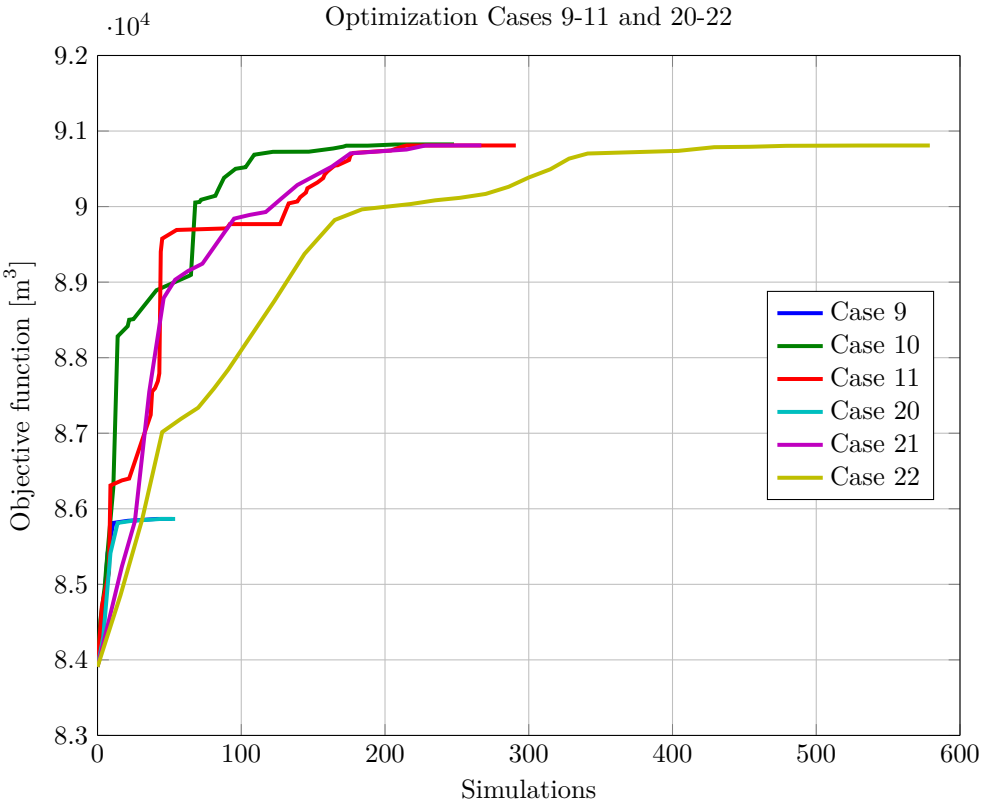

Figure 5.12: Plot of cases 9-11 with Hooke-Jeeves Direct Search and cases 20-22 with Compass Search.

Experiment 2 overall comments Comparing case 8 and 19 shows that it is actually possible to reach the same objective function value while not having the same perforation configuration. The most probable reason for this is that the different perforation configurations are placed within the same grid block. The simulator does not take the perforation placement into account, it only calculates the transmissibility from one grid block to another.

The simulated evaluations ratio for cases 16-22 spans from 57 to 75 %. This is in contrast to the respective cases run with Hooke-Jeeves direct search (cases 5-11) with ratio ranging from 60 to 83 %. This can be seen as a clear effect of the way these two algorithms creates new candidate configurations. The compass search evaluates all valid candidates in both positive and negative directions, while Hooke-Jeeves direct search only evaluate in the negative direction if the positive direction does not find a better objective function value. The Hooke-Jeeves algorithm will never suggest many of the candidate solutions that the compass search will take into evaluation. Based on what we know from the implementation of the two algorithms, the different ratio results for these two algorithms is expected.

5.4 Conclusion

Reduction of the tolerance resulted in more simulations, but not necessarily a better objective function value. When tolerance becomes smaller, the step length towards the end of the optimization run also gets smaller. The new candidate configurations at this time will therefore not be much different than the current best case. This is probably one of the reasons why we do not get much better results when reducing the tolerance to such a low value as 0.5 meter. Another reason may be that the grid blocks in the simulation case is large. If they are much larger than the step length, new perforation configurations will not change the simulation results. The simulator will not notice if a perforation change location within a grid block.

We find that the change in initial step length affects where in the search space we start our search. This parameter is very important for how the optimization run progresses and where in the search space it finishes. We cannot in advance tell what is the best initial step length when using these algorithms, because this parameter is highly dependent on the simulation case.

By testing different segment resolutions these experiments gave the highest objective function value for case 5 with 4 segments. We cannot conclude that this is the best resolution for this simulation case, because cases run with 8 and 16 segments reach objective function values that are almost the same. More tests may be run to see if there is any of the segment resolutions that really stands out as the best solution. This parameter may also be very case sensitive because we do not know how different simulation cases will react to these resolutions. The increase to 16 segments (32 control variables) indicated more perforation coverage at the heel and toe of the well. The toe and heel of the well covers a larger part of the formation. Inflow from the formation will be higher in the toe and heel compared to other parts of the well.

By running optimizations with the same input parameters for Hooke-Jeeves

direct search and compass search we find that compass search requires more simulations to reach the tolerance. As control variables increase this difference becomes larger. This is a consequence of compass search evaluating all directions before analyzing the resulting objective function values. Hooke-Jeeves direct search goes with the first and better solution it finds. When applying this plug-in to more complex simulation cases where each simulation requires more time, the number of simulations will be an important factor. If we compare the best case perforation configurations and objective function values the algorithm differences are so small that they are not worth mentioning. We see that the two algorithms find approximately the same best cases, but Hooke-Jeeves direct search is able to find the best case solution faster than the compass search.

The Hooke-Jeeves direct search can be considered more complex compared to compass search method. While compass search continues to search in the perimeter of the current base point, Hooke-Jeeves exploit the knowledge acquired through exploratory moves and uses this to its advantage. Using the pattern search is a more efficient way to find an optimum in the search space if such an area exists.

We have discussed that both resolution and initial step length may be case dependent. When using this plug-in we should therefore pay most attention to these two parameters. The initial step length provides the algorithms with a starting point for the search, and this has great impact to where in the search space the algorithm finishes. The resolution divides the well into smaller segments and gives us the opportunity to find the best number of perforation intervals for the given well. Having many segments result in many control variables for the optimization run, but can give a more detailed configuration.

Based on the results from these experiments a proposed approach is to structure the optimization runs when applying this plug-in. First, find one or a few resolutions that produce good enough best case objective function values. For these resolutions, investigate more by changing the initial step lengths. See if there is a step length that may find an optimum in the search space. When good values for these two parameters are found, the tolerance can be reduced to find a more detailed configuration.

Chapter 6

Summary and further work

6.1 Summary

Implementation The original Optimization plug-in was extended with a second pattern search method. Both compass search and Hooke-Jeeves can now be used to optimize perforation configurations within a well segment for a given objective function value. Some modifications of the original compass search were implemented for the two algorithms to be properly comparable.

Algorithm comparison It is clear from the total numbers of simulations that Hooke-Jeeves direct search is more efficient than the compass search. The assumption that there may be a better function value in a detected promising direction of the search space saves a lot of simulation runs. Compass search moves very slowly towards the optimum in comparison with Hooke-Jeeves direct search.

Grid block size Both when it comes to initial step length and segment resolution we need to take the grid block size of the simulation case into consideration. If the segment resolution is small with respect to grid block size, change of a perforation within a segment will not be noticed by the reservoir simulator. The change in perforation configuration will therefore not give changes in objective function value. A perforation may also become trapped within a grid block. If the step length is too short to create a new configuration that crosses a grid block interface, change in the perforation configuration will not give any different simulation results.

Different approaches for new perturbations Since Hooke-Jeeves was implemented with evaluating each of the MD variables in both positive and negative direction, the compass search was changed to use this same approach. It would be interesting to see if the two different implementations of compass search that was discussed in chapter 4 find the same best case objective function values and perforation configurations. The total number of simulations will probably also be

different. They will probably not find the same solutions because we have seen that the perforation configurations in these experiments does not seem to be placed in the center of the segments. We conclude that the details of how an algorithm is implemented may have large impact on the result when using the algorithm.

6.2 Recommendations for further work

Perforations on/off In this thesis we have only been concerned with the placement of a perforation within a given segment. What if we shut off one or more of the perforation intervals, e.g. so that only three out of four perforations can produce. It would be interesting to see how this concealment will affect the objective function value.

Optimization of other well completions For this Optimization plug-in to be useful for optimizing the entire well completion process, the plug-in can be extended even more. Other well completion types may be e.g. ICDs, sand control systems, packers, and the different settings for these completions.

More algorithms Currently the plug-in is specialized for pattern search methods. We implemented an `OptimizationAlgorithm` parent class for `CompassSearch` and `HJDirectSearch`. If algorithms which are not of pattern search type is to be implemented, the name of the parent class can be changed to `PatternSearchAlgorithm` to separate the categories.

Additional well It is possible to add another well as input parameter, and optimize perforation configurations for both wells at the same time. If this was implemented, the Optimization plug-in would be able to handle more complicated simulation cases.

Optimization results into file The output from a run with the Optimization plug-in is now done in the Petrel Message Log. If this output was written and automatically saved to a file, we can run several optimizations in sequence using the Petrel workflow editor.

References

[1] Statoil, "2015 annual report on form 20-f," 2015, accessed May 24th 2016. [Online]. Available: http://www.statoil.com/no/InvestorCentre/AnnualReport/AnnualReport2015/Documents/DownloadCentreFiles/01_KeyDownloads/Annual_report_on_form_20-F.pdf.

[2] S. Rassenfoss and A. Henni, "Low oil prices make innovation a priority," *Conference Review, Society of Petroleum Engineers*, February 2015, SPE-0215-0056, DOI: 10.2118/0215-0056-JPT.

[3] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed. Springer, 2006, DOI: 10.1007/978-0-387-40065-5.

[4] R. Hooke and T. A. Jeeves, ""Direct Search" Solution of Numerical and Statistical Problems," *J. ACM*, vol. 8, no. 2, pp. 212–229, Apr. 1961. [Online]. Available: http://doi.acm.org/10.1145/321062.321069

[5] R. M. Lewis, V. Torczon, and M. W. Trosset, "Direct search methods: then and now," *Journal of Computational and Applied Mathematics*, vol. 124, no. 1–2, pp. 191 – 207, 2000, numerical Analysis 2000. Vol. IV: Optimization and Nonlinear Equations. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0377042700004234

[6] T. G. Kolda, R. M. Lewis, and V. Torczon, "Optimization by direct search: New perspectives on some classical and modern methods," *SIAM Review*, vol. 45, no. 3, pp. 385–482, 2003. [Online]. Available: http://dx.doi.org/10.1137/S003614450242889

[7] E. J. M. Baumann, "Fieldopt: Enhanced software framework for petroleum field optimization," Master's thesis, Norwergian University of Science and Technology, June 2015.

[8] M. J. Box, D. Davies, and W. H. Swann, *Non-linear optimization techniques*. Published for Imperial Chemical Industries Ltd by Oliver Boyd, 1969.

[9] T. Ellis, A. Erkal, G. Goh, T. Jokela, S. Kvernstuen, E. Leung, T. Moen, F. Porturas, T. Skillingstad, P. B. Vorkinn, and A. G. Raffn, "Inflow control devices - raising profiles," *Schlumberger Oilfield Review*, vol. 21, no. 4, pp. 30–37, 2010, [Online]. Available: http://69.18.148.100/~/media/Files/resources/oilfield_review/ors09/win09/03_inflow_control_devices.pdf.

[10] S. of Petroleum Engineers, accessed May 13th 2016. [Online]. Available: http://petrowiki.org/PEH%3APerforating.

[11] F. Jahn, M. Cook, and M. Graham, *Hydrocarbon Exploration & Production*, 2nd ed. Elsevier Science, March 2008, iSBN0-444-53236.

[12] M. Avriel, *Nonlinear programming: analysis and methods*. Prentice-Hall, 1976.

[13] Petrel E&P Software Platform by Schlumberger, version 2015.4. Accessed May 23rd 2016. [Online]. Available: https://www.software.slb.com/products/petrel.

[14] ECLIPSE 100 Industry-Reference Reservoir Simulator by Schlumberger. Accessed May 28th 2016. [Online]. Available: https://www.software.slb.com/products/eclipse.

[15] Ocean Software Development Framework by Schlumberger, version 2015. Accessed May 23rd 2016. [Online]. Available: https://www.ocean.slb.com/en.

[16] L. Bass, P. Clements, and R. Kazman, *Software Arcitecture in Practice*, 3rd ed. Addison-Wesley, 2013.

[17] Microsoft., ".NET," accessed May 28th 2016. [Online]. Available: https://www.microsoft.com/net.

[18] Microsoft, "Introduction to the C# Language and the .NET Framework," accessed April 28th 2016. [Online]. Available: https://msdn.microsoft.com/en-us/library/z1zx9t92.aspx.

[19] Microsoft Visual Studio, "Introducing visual studio," accessed May 24th 2016. [Online]. Available: https://www.visualstudio.com/products/vs-2015-product-editions.

[20] I. Sommerville, *Software Engineering*, 9th ed. Pearson, 2011.

[21] H. Trætteberg, "Objectorientert programmering," last edited Oct 10 2013. Accessed: May 27th 2016. [Online]. Available: https://www.ntnu.no/wiki/display/tdt4100/Objektorientert+programmering.

[22] B. Stroustrup, *The C++ programming language*, 4th ed. Addison-Wesley, 2013.

[23] *Petrel Help Center*, Schlumberger, manual for Petrel E&P Software Platform, version 2015.2.

[24] Schlumberger, "Schlumberger oilfield glossary - MD," accessed May 30th 2016. [Online]. Available: http://www.glossary.oilfield.slb.com/Terms/m/md.aspx.

Appendix A

Experiment results

Segments	StartMD	EndMD	Base case objective function value
2	1095.31	1393.47	83908.11
	1393.47	1691.64	
4	1095.31	1244.39	83908.11
	1244.39	1393.47	
	1393.47	1542.56	
	1542.56	1691.64	
8	1095.31	1169.85	83908.11
	1169.85	1244.39	
	1244.39	1318.93	
	1318.93	1393.47	
	1393.47	1468.02	
	1468.02	1542.56	
	1542.56	1617.10	
	1617.10	1691.64	
16	1095.31	1132.58	83908.11
	1132.58	1169.85	
	1169.85	1207.12	
	1207.12	1244.39	
	1244.39	1281.66	
	1281.66	1318.93	
	1318.93	1356.20	
	1356.20	1393.47	
	1393.47	1430.74	
	1430.74	1468.02	
	1468.02	1505.29	
	1505.29	1542.56	
	1542.56	1579.83	
	1579.83	1617.10	
	1617.10	1654.37	
	1654.37	1691.64	

Table A.1: Base case perforation configurations for segments 2, 4, 8 and 16 with full coverage.

Case no.	StartMD	EndMD	Best case objective function value
Case 1	1095.31	1393.47	85840.77
	1505.97	1691.64	
Case 2	1095.31	1393.47	85840.77
	1505.97	1691.64	
Case 3	1095.31	1393.47	85865.46
	1503.16	1691.64	
Case 4	1095.31	1393.47	85866.96
	1503.86	1691.64	
Case 5	1096.71	1143.14	90823.04
	1316.11	1337.22	
	1494.72	1497.56	
	1590.37	1633.98	
Case 6	1095.31	1146.89	90818.74
	1315.64	1337.22	
	1505.97	1514.43	
	1595.06	1635.39	
Case 7	1095.31	1144.39	90817.97
	1308.45	1318.47	
	1405.97	1417.56	
	1589.43	1635.39	
Case 8	1095.31	1144.39	90822.66
	1303.14	1313.47	
	1400.97	1412.56	
	1590.06	1636.64	
Case 9	1095.31	1393.47	85865.46
	1503.16	1691.64	
Case 10	1096.71	1143.14	90823.04
	1316.11	1337.22	
	1494.72	1497.56	
	1590.37	1633.98	
Case 11	1095.31	1147.35	90808.96
	1242.97	1244.39	
	1309.08	1318.93	
	1358.31	1370.97	
	1466.60	1468.02	
	1541.14	1542.56	
	1590.37	1600.22	
	1638.19	1674.76	

Table A.2: Best case perforation configurations for cases 1 - 11 using Hooke-Jeeves Direct Search.

Case no.	StartMD	EndMD	Best case objective function value
Case 12	1095.31	1391.60	83926.98
	1393.47	1691.64	
Case 13	1095.31	1146.89	90819.70
	1308.14	1318.47	
	1395.35	1407.56	
	1589.43	1635.39	
Case 14	1095.31	1143.60	90820.00
	1211.10	1214.39	
	1289.39	1300.18	
	1369.56	1374.72	
	1408.47	1415.52	
	1509.27	1512.56	
	1591.31	1598.35	
	1635.85	1672.89	
Case 15	1095.31	1132.58	90819.39
	1132.58	1143.60	
	1190.47	1192.12	
	1227.75	1229.39	
	1265.02	1266.66	
	1306.04	1315.18	
	1339.56	1341.20	
	1376.83	1378.47	
	1406.60	1415.74	
	1451.37	1453.02	
	1488.64	1490.29	
	1525.91	1527.56	
	1578.18	1579.83	
	1592.95	1605.85	
	1628.35	1643.12	
	1665.62	1682.27	

Table A.3: Best case perforation configurations for cases 12 - 15 using Hooke-Jeeves Direct Search.

Case no.	StartMD	EndMD	Best case objective function value
Case 16	1095.31	1143.14	90809.89
	1314.70	1338.63	
	1531.29	1542.56	
	1655.06	1691.64	
Case 17	1095.31	1146.89	90820.34
	1315.64	1337.22	
	1532.22	1542.56	
	1655.06	1691.64	
Case 18	1096.87	1144.39	90821.37
	1316.27	1337.22	
	1532.54	1542.56	
	1658.18	1691.64	
Case 19	1095.31	1144.39	90822.66
	1313.14	1323.47	
	1400.97	1412.56	
	1590.06	1636.64	
Case 20	1095.31	1393.47	85865.46
	1503.16	1691.64	
Case 21	1095.31	1143.14	90809.89
	1314.70	1338.63	
	1531.29	1542.56	
	1655.06	1691.64	
Case 22	1095.31	1147.35	90808.66
	1240.16	1244.39	
	1311.89	1318.93	
	1387.84	1393.47	
	1393.47	1400.52	
	1538.33	1542.56	
	1587.56	1594.60	
	1655.07	1691.64	

Table A.4: Best case perforation configurations for cases 16 - 22 using Compass Search.