NTNU

Norwegian University of
Science and Technology

On the Importance of Expert Knowledge in Well Placement Optimisation

Alexander Zakharov

Petroleum Geoscience and Engineering
Submission date: June 2016
Supervisor: Jon Kleppe, IPT
Co-supervisor: Mathias Bellout, IPT

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

Abstract

As of today, well placement optimisation in field development planning is most commonly based on the trial-and-error approach that requires the evaluation of numerous field development scenarios. This approach is not particularly efficient because it requires a lot of repetitive work to be done by reservoir engineers. Moreover, such approach can lead to overlooked opportunities and suboptimal well placement, which, in turn, can have a negative economic impact on a project in the long-term perspective.

This work describes an effort to better understand the role of reservoir engineering expert knowledge in well placement optimisation at the stage of field development planning and figure out whether an appropriate choice of optimiser parameters can help reduce the dependence on experience-based knowledge. The research question is thus formulated as:

What is the role of expert knowledge in the process of well placement optimisation, and how to reduce the dependence on such knowledge?

An experimental approach is taken. Numerical experiments are conducted on a two-dimensional reservoir model to understand how the quality of initial approximation and two main optimiser parameters affect the optimisation results.

A purpose-specific optimiser-simulator interface is developed to integrate the HOPSPACK optimiser with the ECLIPSE reservoir simulator.

The interpretation of numerical experiments leads to the following two major conclusions for this particular two-dimensional optimisation problem:

- Reservoir engineering expert knowledge plays an important role in the process of well placement optimisation by providing a good quality initial approximation;
- A proper choice of the optimiser parameters that allow for a wide exploration of the search space may significantly reduce the dependence on such expert knowledge.

It is demonstrated that for the relatively simple reservoir model considered in this study, optimisation techniques can enhance the well placement optimisation workflow. Further research focused on more complex three-dimensional reservoir models may contribute to dispelling the general scepticism towards the use of optimisation techniques for real life situations.

Sammendrag

Nåværende praksis i optimalisering av brønnplassering i feltutviklingsplanlegging er oftest basert på en prøve-og-feile tilnærming, som krever en vurdering av mange feltutviklingsscenarier. Denne tilnærmingen er ikke spesielt effektiv, fordi den krever at reservoaringeniørene må gjøre mye repeterende arbeid. Dessuten kan en slik tilnærming føre til oversette muligheter og suboptimale brønnplasseringer, som kan ha negative konsekvenser for et prosjekt i et langsiktig perspektiv.

Dette arbeidet undersøker hvordan rollen av reservoarteknisk ekspertkunnskap i optimalisering av brønnplassering i feltplanleggingsfasen er og om et passende valg av optimalisator parametre kan bidra til å redusere avhengigheten av erfaringsbasert kunnskap. Forskningsspørsmålet er derfor formulert som:

Hva er rollen til ekspertkunnskap i prosessen med å optimalisere brønnplasseringer, og hvordan kan man redusere avhengigheten av slik kunnskap?

En eksperimentell tilnærming er tatt. Numeriske eksperimenter ble utført på en todimensjonal reservoarmodell for å forstå hvordan kvaliteten på første antagelse og to hovedoptimalisator parametre påvirker optimaliseringsresultatene.

Et tilpasset optimalisator-simulator grensesnitt er utviklet for å integrere HOPSPACK optimalisator sammen med ECLIPSE reservoarsimulator.

Tolkningen av de numeriske eksperimentene fører til følgende to hovedkonklusjoner:

- Reservoarteknisk ekspertkunnskap spiller en viktig rolle i prosessen med å optimalisere brønnplasseringen, ved å tilby en første antagelse av god kvalitet;
- Et riktig valg av optimalisator parametre legger til rette for en bred utforskning av søkeområdet, og kan betydelig redusere avhengighet av slik ekspertkunnskap.

Det er vist at for den relativt enkle reservoarmodellen vurdert i denne studien, kan optimaliseringsteknikker forbedre arbeidsflyten knyttet til optimalisering av brønnplasseringer. Videre forskning med fokus på mer komplekse tredimensjonale reservoarmodeller kan bidra til å dempe den generelle skepsisen til bruk av optimaliseringsteknikker i virkelige situasjoner.

Acknowledgements

First and foremost, I would like to thank my main supervisor, Prof. Jon Kleppe, and my co-supervisor, Postdoc Mathias Bellout, for providing insightful comments and helping me find a sense of direction in my work. It has been a privilege working with them.

Many thanks go to all my friends at NTNU. I consider myself lucky to be surrounded by great friends, who have supported and motivated me throughout my academic career.

Finally, I would like to express my sincere gratitude to my family for their constant encouragement, love and support, without which I certainly would not be where I am today.

Abbreviations

APPS	–	Asynchronous Parallel Pattern Search
APPSPACK	–	Asynchronous Parallel Search PACKage
CF	–	Contraction Factor
GPS	–	Generalised Pattern Search
HOPSPACK	–	Hybrid Optimisation Parallel Search PACKage
MPI	–	Message Passing Interface
PSO	–	Particle Swarm Optimisation
SP	–	Starting Point
SS	–	Step Size

Table of Contents

Abstract i

Sammendrag iii

Acknowledgements v

Abbreviations vii

Table of Contents ix

List of Tables xi

List of Figures xi

1 Introduction 1

2 Literature Review 3
- 2.1 Classification of Optimisation Methods . . . 3
- 2.2 Generalised Pattern Search . . . 4
- 2.3 Asynchronous Parallel Pattern Search . . . 5
- 2.4 Particle Swarm Optimisation . . . 7
- 2.5 Algorithm Chosen for Numerical Experiments . . . 8

3 Methodology and Tools 9
- 3.1 Reservoir Model Description . . . 9
- 3.2 HOPSPACK Optimiser . . . 10
- 3.3 Optimiser-Simulator Interface . . . 11
- 3.4 Independent Variables . . . 11
- 3.5 Data Analysis Methodology . . . 13

4 Experimental Results 15
- 4.1 Summary of Experimental Results . . . 15
- 4.2 Significance of Initial Well Placement . . . 17
- 4.3 Significance of Optimiser Parameters . . . 19
  - 4.3.1 Initial Step Size . . . 20
  - 4.3.2 Contraction Factor . . . 21
- 4.4 Optimal Well Placement . . . 22

5 Discussion 25
- 5.1 Interpretation of Numerical Experiments . . . 25
- 5.2 Guiding Principles for Optimal Well Placement . . . 27
- 5.3 Future Work . . . 28

6 Conclusion 29

Bibliography 31

Appendices 32

List of Tables

1 Summary of independent variables and their purpose 12
2 Best-case scenario well coordinates 22

List of Figures

1 Griewank function [3] 3
2 Convergence of GPS to a stationary point [6] 5
3 Eight iterations of APPS [11] 6
4 Illustration of particle's velocity and position update [8] 7
5 Reservoir grid properties 10
6 Optimiser-simulator interface: single iteration procedure 11
7 Conceptual illustration of the different choices for three variables . . 13
8 Ultimate recovery improvement over progressing iterations 15
9 Ultimate recovery and number of iterations for each simulation run 16
10 Step size (SP5) evolution with different contraction factors 17
11 Influence of initial well placement on ultimate recovery 17
12 Comparison of different well placement strategies 18
13 Occurrence count at particular threshold ultimate recovery 18
14 Influence of the optimisation parameters on ultimate recovery . . . 19
15 Influence of initial step size on average ultimate recovery 20
16 Influence of contraction factor on average ultimate recovery 21
17 Initial and optimal well locations 23
18 Initial and optimal final oil saturations 24
19 HOPSPACK configuration file 32
20 HOPSPACK configuration parameters for SP1 scenarios 33
21 HOPSPACK configuration parameters for SP2 scenarios 34
22 HOPSPACK configuration parameters for SP3 scenarios 35
23 HOPSPACK configuration parameters for SP4 scenarios 36
24 HOPSPACK configuration parameters for SP5 scenarios 37
25 HOPSPACK configuration parameters for SP6 scenarios 38
26 HOPSPACK configuration parameters for SP7 scenarios 39

1 Introduction

For decades, petroleum engineers have played a major role in oil and gas field development activities. Their expert knowledge is considered indispensable when it comes to making the most important field development decisions, which predetermine ultimate oil and gas recovery and thereby have a considerable economic impact. Planned well count and well placement[1]are examples of such decisions.

In the simplest cases, well placement patterns can be chosen quite intuitively. However, optimisation of well placement turns into a non-trivial problem when it comes to reservoirs with complex geometry and heterogeneous property distribution.

As of today, well placement optimisation is most commonly based on a mathematical modelling of the numerous field development scenarios using reservoir simulators and the trial-and-error approach. This approach is not particularly efficient because it requires a lot of repetitive work to be done by reservoir engineers. Besides, the current practice can easily lead to overlooked opportunities and suboptimal well placement which, in turn, can have a negative economic impact on a project in the long-term perspective.

From a mathematical standpoint, well placement optimisation is considered to be a challenging problem for at least three reasons.

Firstly, the reservoir system is not very well defined at the stage of field development planning. Indeed, the true, complete and deterministic information about the reservoir is never available in advance prior to massive development drilling. Therefore, this optimisation problem is usually being solved using a reservoir model constructed with the limited amount of information available to date and based on one of many possible geostatistical realisations [1].

Secondly, reservoir heterogeneity along with the complexity of its geometry dramatically increase the number of optimisation variables and inflate the search space.

Thirdly, the objective function for such systems becomes non-smooth and multimodal. Most known optimisation methods would struggle to find the global optimum.

For these reasons, the best work efficiency can be achieved when mathematical and reservoir engineering knowledge is combined together for solving the well placement optimisation problem.

Engineers can offer their reservoir understanding and expertise to greatly assist with the search of optimal well placement solution by providing a good initial approximation (i.e. tentative well placement configuration to start with). Otherwise, a path from the initial approximation to the optimal well placement solution becomes tortuous, making calculations enormously time consuming and computer intensive at best, or making the algorithm non-convergent at worst.

[1]The term "well placement" in this work refers to planning of irregular well patterns for oil and gas fields. In the context of this work it has nothing to do with the well placement optimisation problem being solved while drilling based on real-time logging data.

In this context, reservoir engineering expert knowledge and understanding of the bigger picture are considered invaluable. This may be the main reason why computers have not yet replaced engineers.

No doubt, a mature engineer with many years of experience can make a great difference in any well placement optimisation project. But what if such resource is not available for the project? Can a less experienced team find an equally good solution to the well placement problem if they use one of the advanced optimisation software packages?

This thesis describes an effort to better understand the role of reservoir engineering expert knowledge in well placement optimisation and figure out whether an appropriate choice of optimiser parameters can help reduce the dependence on such theoretical and experience-based knowledge. The research question is thus formulated as:

What is the role of expert knowledge in the process of well placement optimisation, and how to reduce the dependence on such knowledge?

The well placement optimisation problem is being solved using an advanced optimisation software package.

It is hypothesised that expert knowledge plays an important role in well placement optimisation through a provision of a good initial approximation (i.e. near-optimal initial well placement locations to start iterations with). It is also hypothesised that an appropriately configured optimisation algorithm can significantly reduce the need for prior expert knowledge when well placement problems are being solved.

2 Literature Review

There is a growing interest in application of optimisation techniques to the well placement problem in recent years [2]. These techniques are introduced into well placement optimisation workflows in order to improve current engineering solutions and aid reservoir engineers in the decision-making process.

There is a multitude of different optimisation algorithms. Each algorithm has unique properties and limitations, which make it useful for solving particular optimisation tasks. Most of these optimisation algorithms are classified as either gradient-based or derivative-free.

2.1 Classification of Optimisation Methods

Gradient-based optimisation methods take advantage of gradient information of an objective function in order to determine the optimal search direction (steepest descent). Such methods are particularly efficient at finding local minima for high-dimensional convex problems [3]. However, most gradient optimisers have problems dealing with noisy and discontinuous functions. Moreover, they are not designed to handle multimodal or discrete problems, such as the case with the well placement problem [3].

For example, a gradient-based method would struggle optimising the Griewank function (shown in Figure 1). The Griewank function looks deceptively smooth when plotted in a large domain. However, its irregularity and a vast number of local optima become prominent in a smaller domain.

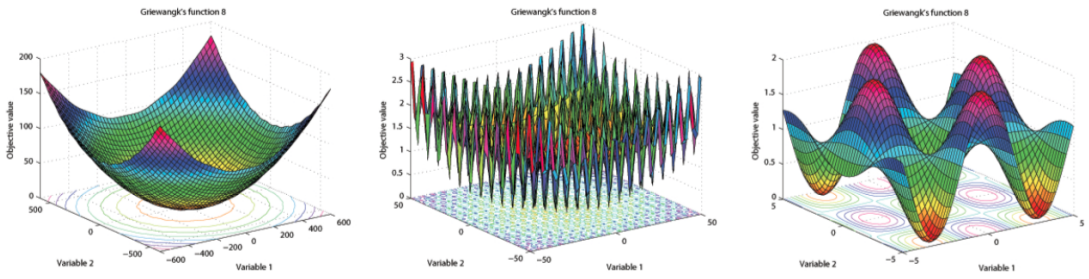


Figure 1: Griewank function [3]

Gradient-based methods are unlikely to find a global optimum of such a function, because they are not designed to handle multimodal problems. Unlike derivative-free methods, gradient-based techniques have no way of escaping local optima. Besides, gradients for complex objective functions may not always be readily available and/or may be difficult to compute.

Gradients provide explicit information about the location of a local optimum, which is why it is considered good practice to utilise such information whenever possible. Such explicit knowledge is the reason why gradient-based methods can outperform derivative-free methods in terms of computational efficiency.

Clearly, gradient-based methods are not well suited for solving the well placement optimisation problem. Firstly, the objective function for such problems is

highly non-smooth and multimodal [4]. Secondly, well placement optimisation is a discrete problem, which makes it impossible to compute gradients. Finally, a reservoir simulator is predominantly treated as a black box. Therefore, extraction of gradients is a simulator invasive process.

Derivative-free methods do not suffer from the same problems, which is why they are often preferred to gradient-based methods for solving the well placement optimisation problem.

Derivative-free (non-invasive black box) optimisation has lately received considerable attention within the optimisation community [5]. Unlike gradient-based methods, derivative-free techniques rely solely on the values of the objective function to guide the search. Derivative-free algorithms employ either a deterministic or a heuristic search strategy. In most cases, it means deploying a population of agents that sample the objective function at different locations. The agents share best-known objective function values amongst themselves through broadcasting. This information is then used to guide the search and converge to a final solution.

Most derivative-free methods can be classified as either deterministic or stochastic techniques. Deterministic techniques use a set of rules to explore the search space and are usually associated with local search. Unlike deterministic techniques, stochastic techniques are associated with a certain degree of randomness in the search process. Most stochastic techniques are designed for a global exploration of the search space.

For the aforementioned reasons, derivative-free methods are the preferred choice for solving the well placement optimisation problem. They are better at dealing with noisy, multimodal and discrete problems. A number of recent publications (see [4, 6–8]) in recent years describe cases where derivative-free optimisation methods were successfully applied for sovling the well placement optimisation problem.

The following subsections will introduce three optimisation algorithms that have been applied to the well placement optimisation problem, namely generalised pattern search (GPS), asynchronous parallel pattern search (APPS) and particle swarm optimisation (PSO).

2.2 Generalised Pattern Search

Generalised pattern search (GPS) refers to a family of numerical optimisation methods that can solve optimisation problems without derivative information. GPS is a local, deterministic search method that samples the objective function over a predefined pattern of points, all of which lie on a rotational lattice [9].

The algorithm explores the search space by polling and evaluating data points in a stencil-based fashion. Having evaluated the data points, the algorithm follows a set of simple search and update instruction that describe how to translate and scale the stencil in order to converge to a stationary point (solution). The process of convergence to a stationary point (solution) is illustrated in Figure 2.

The dominant computational cost for pattern search methods lies in the evaluation of the objective function [9]. Consequently, the first main advantage of the GPS algorithm is that it can be easily implemented in a distributed computing

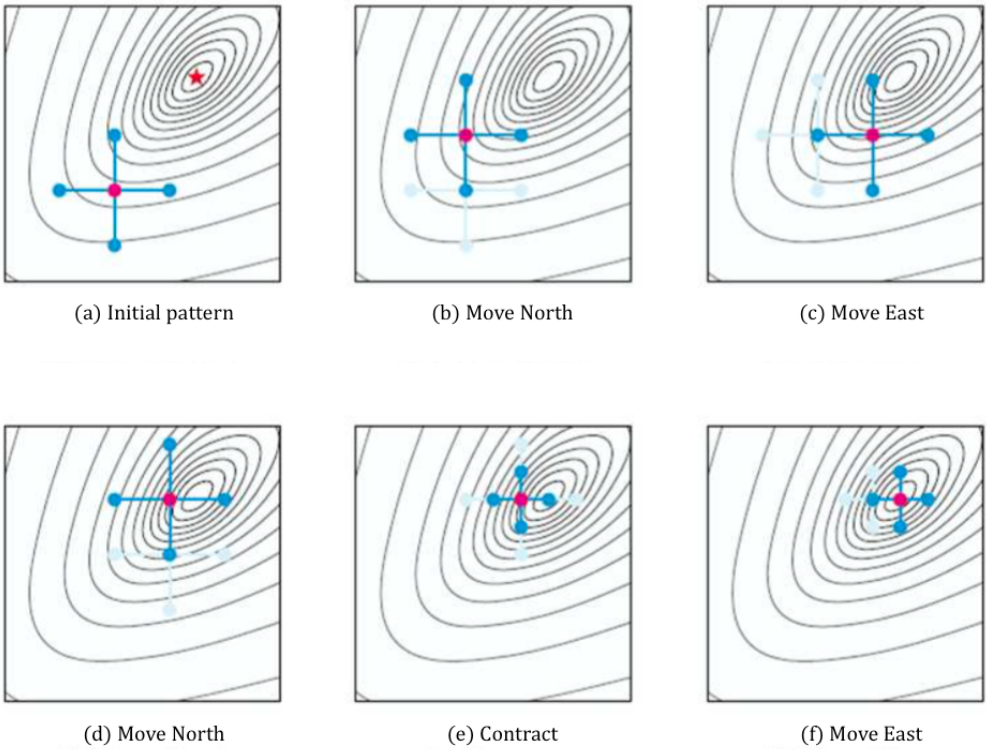


Figure 2: Convergence of GPS to a stationary point [6]

environment. This is achieved by distributing the workload (objective function evaluations) across available computing nodes.

Another advantage is that pattern search methods are supported by a mathematical convergence theory [10]. In fact, this is the reason behind recent popularity of pattern search methods. There is, however, no guarantee that the algorithm will find the true global optimal solution.

The main drawback of the GPS algorithm is that it appears to be quite sensitive to the quality of initial approximation. This means that the algorithm may potentially converge to different solutions under different initial conditions. This may seem as a mild inconvenience when GPS deals with relatively small problems, because the algorithm may be easily run multiple times with different initial approximations. However, this mild inconvenience turns into a large problem when a single iteration is computationally expensive, such as in reservoir simulation.

2.3 Asynchronous Parallel Pattern Search

Traditionally, parallel implementations of pattern search methods are based on the assumption that all objective function evaluations take approximately the same amount of time. Although this assumption may be true for simple problems, it does not always apply in practice [9]. Some optimisation problems are based on complex physical simulations with varying runtimes. Implementations of the pattern search algorithm that are based on the aforementioned assumption usually

suffer from inefficient processor utilisation [9].

This motivated the development of the asynchronous parallel pattern search (APPS) algorithm to make the use of computation cores more efficient. Figure 3 borrowed from [11] shows how APPS explores a larger portion of the search space by decoupling individual function evaluations and finds a solution in 8 iterations in this particular case. In a similar situation, the conventional parallel pattern search algorithm would require more than eight iterations to find the optimum, when the initial approximation is so far from it. For more information about the APPS algorithm refer to [9], [11] and [12].

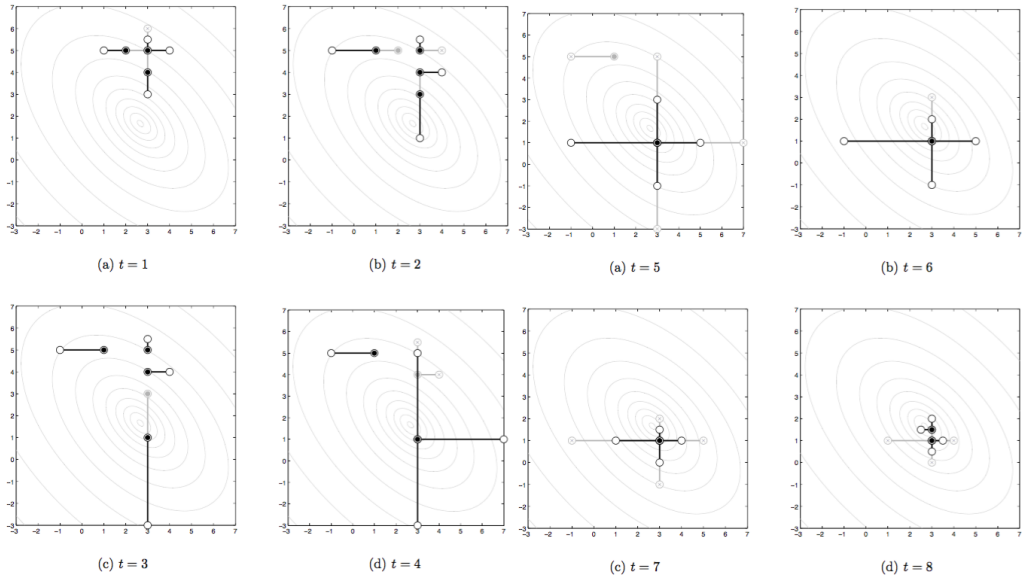


Figure 3: Eight iterations of APPS [11]

The APPS algorithm is programmed in a way that allows it to effectively distribute workload across all available processors. This increases the efficiency of processor utilisation by means of reducing their idle time [9]. The algorithm launches several processes at once and proceeds with the search without waiting for all processes to finish by skipping a synchronisation step. In an asynchronous mode, each process makes decisions based on its best local objective function value, which is then broadcast to other active processes.

Just like any other pattern search method, the APPS algorithm is supported by a solid mathematical convergence theory [10, 12]. This means that the algorithm will converge to a stationary point independent of the initial guess. However, this stationary point is not guaranteed to be a global optimum.

2.4 Particle Swarm Optimisation

Real-world problems usually involve a large number of optimisation variables. This often leads to a high-dimensional search space and a multimodal objective function. Local deterministic optimisation algorithms are not always able to find the true global optimal solution to such problems [10]. Besides, there is no way of recognising a global optimum, unless an exhaustive search is conducted.

The development of global stochastic optimisation algorithms was inspired by the need to address these challenges. These algorithms are far better at exploring a larger portion of the search space and are not as easily trapped in local optima.

Particle swarm optimisation (PSO) is an example of a stochastic global optimisation algorithm. This population-based algorithm is an abstraction of a natural process that mimics social behaviours exhibited by swarms of animals [8]. The optimisation in PSO is brought about by a cooperative search strategy where particles interact with each other. Different interaction patterns are achieved using different neighbourhood topologies, where a particle can interact with a certain number of its neighbouring particles [8].

There are two extremes in PSO topologies. One that focuses on a purely local search (few neighbouring connections) and the other that focuses on a global search (all particles are connected). Most variants of topologies fall somewhere between the two extremes. The number of neighbouring connections determines the balance between local and global exploration. The performance of the algorithm may highly depend on the choice of topology.

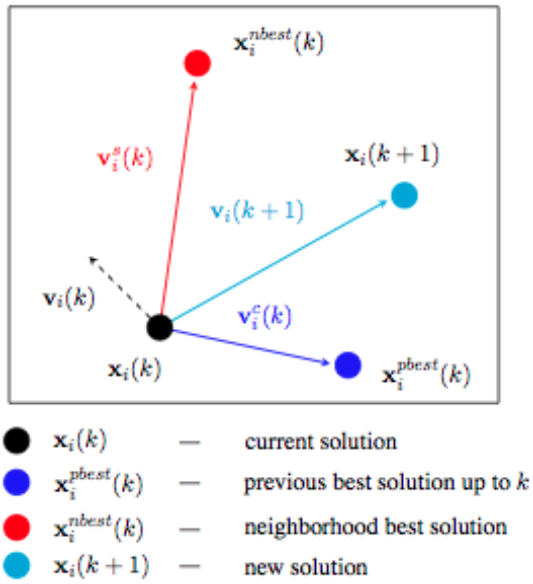


Figure 4: Illustration of particle's velocity and position update [8]

The concept behind PSO algorithm is extremely simple. There is a swarm of particles (solutions) that explore the search space and interact with each other in order to find an optimum. The position and velocity of each particle are updated iteratively according to the objective function value [8] as shown in Figure 4. The

stochastic nature of this algorithm manifests itself as random coefficients that affect particle velocity. The velocity of a particle is partially influenced by the information from other particles. The importance of this information is affected by neighbourhood topology and the aforementioned random coefficients.

The beauty of this algorithm is that it is extremely simple to implement, requiring only the update of particle's velocity and position. Furthermore, the algorithm has few adjustment parameters. This makes PSO a very robust algorithm compared to other stochastic optimisation algorithms that depend on the choice of correct parameters, such as the genetic algorithm [13]. Finally, PSO does not require pre-processing or encoding of input data, as is the case with some other methods.

In spite of the fact that PSO is quite a robust method, its performance depends on the values assigned to the optimiser parameters. The main drawback of PSO is that it lacks a solid mathematical convergence theory. The two problems, however, can be addressed by an overlaying optimiser, a concept known as meta-optimisation [14].

2.5 Algorithm Chosen for Numerical Experiments

Asynchronous parallel pattern search (APPS) algorithm is used in this study for solving the well placement optimisation problem. This particular algorithm is chosen for its novelty, advanced nature and computational efficiency. As mentioned above, it belongs to a larger group of pattern search methods that are backed by a mathematical convergence theory.

Such choice is also inspired by a successful application of HOPSPACK (an implementation of the APPS algorithm) to the well placement optimisation problem described in [4].

3 Methodology and Tools

An experimental approach is taken. Multiple numerical experiments are conducted to explore the importance of a good initial well placement approximation based on engineering knowledge, on one hand, and an appropriate choice of the two most important optimiser parameters, on the other hand.

Optimisation experiments are designed to figure out whether a better choice of optimiser parameters can compensate for the shortage of reservoir engineering expert knowledge, or not. The latter is materialised in a suboptimal well placement configuration at the starting point. These "no prior expert knowledge" scenarios are benchmarked against the "expert" scenarios with a near-optimal initial well placement.

The significance of optimiser settings is investigated by altering two main search parameters of the optimiser. The first parameter controls the algorithm's explorative property (i.e. width of search), whereas the second parameter controls its speed of convergence. By testing a range of different optimiser settings, it would be possible to conclude whether there is a preferred way to configure the optimiser.

Numerical experiments for the purpose of this study are conducted on a two-phase two-dimensional heterogeneous reservoir model described in Subsection 3.1 below.

The following software tools are used:

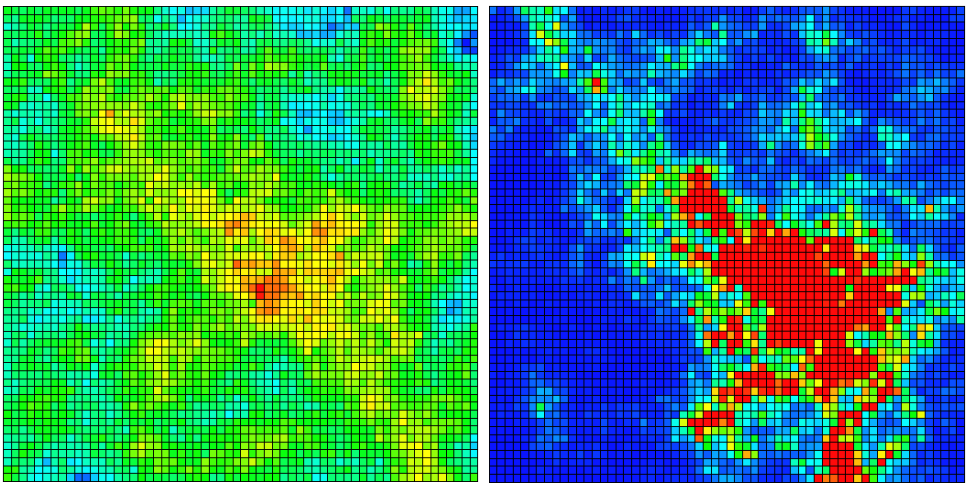
- ECLIPSE E100 Black Oil reservoir simulator owned by Schlumberger;
- HOPSPACK optimisation software package developed by Sandia National Laboratories that is based on an asynchronous parallel pattern search (APPS) algorithm.

3.1 Reservoir Model Description

The reservoir model used for numerical experiments is a two-phase (oil and water) two-dimensional (2-D) model that has a simple geometry with no dipping or faults. The model is defined on a regular Cartesian grid measuring $60\times60\times1$ cells (total of 3600 cells). The dimensions of the model are $1440\times1440\times24$ metres, with a uniform size for each block being $24\times24\times24$ metres. The top of the model is located at a depth of 1700 metres. Initial pressure at this depth is 170 barsa. The model is a cut-off of layer 21 of the SPE 10 model [15].

Property distribution in the reservoir model is heterogeneous. The porosity and horizontal permeability distributions are shown in Figure 5. Permeability in X- an Y directions is assumed to be the same.

Four producers and one water injector are included in the reservoir model. All wells are controlled by a fixed bottomhole pressure at a reference depth of 1715 metres. The bottomhole pressure for the oil producers is set to 90 barsa and the bottomhole pressure for the injector is set to 230 barsa. 12 years of oil production with water injection are simulated with a maximum time step of 73 days.



(a) Porosity (b) Horizontal permeability

Figure 5: Reservoir grid properties

Despite an apparent simplicity of the model, optimisation of well placement in it is considered to be a mathematically complex problem due ten degrees of freedom (namely five wells, each having X and Y coordinates as optimisation parameters). In theory, reservoir heterogeneity and a high degree of freedom lead to a highly non-smooth multimodal objective function [4] that is very difficult to optimise.

3.2 HOPSPACK Optimiser

The optimisation software package used in the numerical experiments is HOPSPACK (Hybrid Optimisation Parallel Search PACKage). HOPSPACK is a successor to Sandia National Laboratory's APPSPACK (Asynchronous Parallel Pattern Search PACKage) product. It builds on the last version of APPSPACK and extends its capabilities.

HOPSPACK is a derivative-free optimisation software for solving general optimisation problems, especially those with noisy and computationally expensive functions. It is implemented in C++ programming language and supports parallel operations using MPI (Message Passing Interface) or multithreading. HOPSPACK allows variables to be continuous or integer-valued and has provisions for multi-objective optimisation problems [16].

Key features of HOPSPACK:

- Only function values are required for the optimisation (no derivatives)
- The user only needs to provide a program that can evaluate the objective function at a given point
- HOPSPACK can be run in parallel
- An asynchronous implementation of the Generating Set Search (GSS) algorithm is supplied, which is a type of pattern search solver.

In our case, HOPSPACK is configured to solve a single-objective optimisation problem. The objective function is the cumulative oil production at the end of the reservoir simulation. This objective function has to be maximised by HOPSPACK.

3.3 Optimiser-Simulator Interface

In order to use HOPSPACK with ECLIPSE reservoir simulator, a purpose-specific optimiser-simulator interface is developed as a part of this project. Namely, a function evaluator executable is written in C++ programming language. The functionality of this executable is to receive input data from HOPSPACK, modify well data (WELSPECS and COMPDAT) in ECLIPSE data deck, launch ECLIPSE, read the resulting cumulative oil production from a summary file, feed it as an objective function value back to HOPSPACK and keep a record of the iteration results. The procedure is summarised in Figure 6. HOPSPACK configuration file and optimiser-simulator interface code can be found in Appndices A and C, respectively

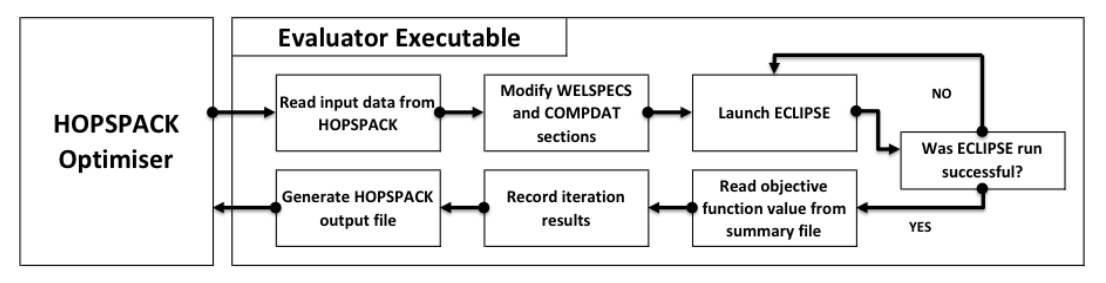


Figure 6: Optimiser-simulator interface: single iteration procedure

3.4 Independent Variables

The optimisation experiments in this study are conducted with three independent variables, namely initial well placement, initial step size and stencil contraction factor. Information about the independent variables is summarised in Table 1.

The purpose of introducing seven different choices for the initial well locations is to model situations with various degree of expert knowledge use. Initial placement

Table 1: Summary of independent variables and their purpose

Independent Variable	**Number of Choices**	**Purpose**
Initial well placement	7	Demonstrate the importance of near-optimal well placement (hence expert knowledge)
Initial step size	5	Demonstrate how controlling the width of search affects optimisation results
Stencil contraction factor	5	Demonstrate how controlling the convergence speed affects optimisation results

of the wells in their near-optimal location corresponds to the case with the most extensive use of the expert knowledge in this optimisation problem. Conversely, initial placement of the wells in their most suboptimal location corresponds to the case with no expert knowledge use.

Supposedly, placing wells in their near-optimal location would help the optimiser converge to a better solution faster as opposed to placing wells in a suboptimal location.

Well's location is a subject to a constraint. To minimise the boundary effects, it is decided that there has to be a minimum of five cells between a well and each edge of the grid.

In summary, the initial well placement variable is introduced to demonstrate the importance of near-optimal initial well placement (and hence expert knowledge) in solving the well placement optimisation problem.

Initial step size and stencil contraction factor are the two most important HOPSPACK optimiser parameters.

Initial step size controls the width of search. In theory, a larger step size allows for a more global exploration of the search space, while a smaller step size mimics a local search.

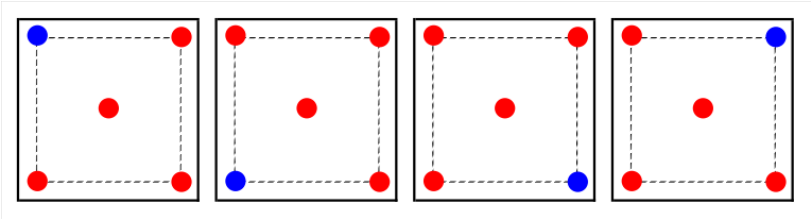
Stencil contraction factor affects the speed of convergence by controlling how fast the stencil reduces in size when it approaches a stationary point. Higher contraction factor lets the stencil shrink at a lower rate, thereby reducing the risk of premature convergence.

It is expected that proper adjustment of the two optimiser parameters can compensate for suboptimal initial well placement.

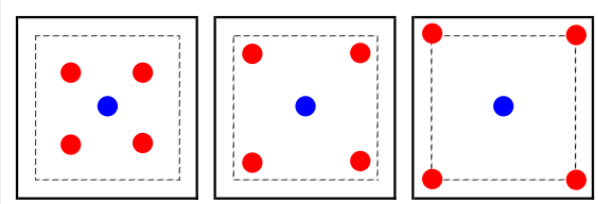
Running multiple cases with different choices and combination of the optimiser parameters would supposedly reveal whether it is possible to reduce the dependence of the well placement optimisation on expert knowledge for provision of a good initial well placement approximation.

With seven possible initial well placement configurations, five initial step sizes (10, 20, 30, 40 and 50) and five contraction factor choices (0.5, 0.6, 0.7, 0.8 and 0.9) in this study, the total number of all their possible combinations is 175. Accordingly, 175 optimisation scenarios are run in this experimental study in order to understand how all the three chosen variables affect the optimisation results.

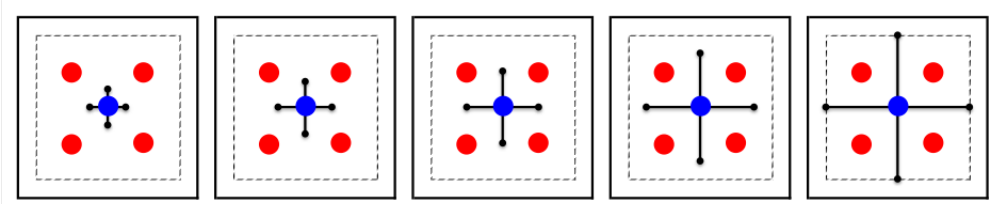
Different choices of the three chosen variables are conceptually illustrated in Figure 7. See also Appendices B1-B7. Production and injection wells are represented by red and blue dots, respectively. The dashed square represents the constraint imposed on well locations (i.e. all the wells have to be located within the dashed square).



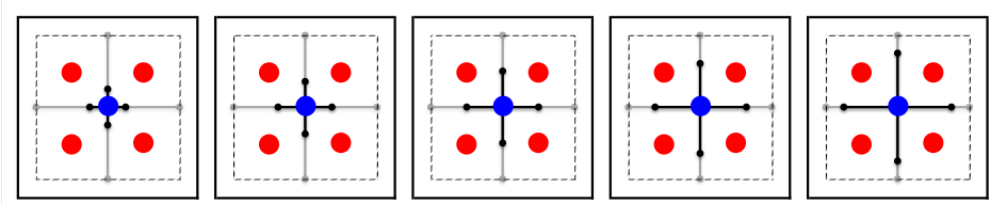
(a) Suboptimal initial well configurations (SP1, SP2, SP3, SP4)



(b) Near-optimal initial well configurations (SP5, SP6, SP7)



(c) Initial step sizes to control width of exploration (SS1, SS2, SS3, SS4, SS5)



(d) Contraction factors to control speed of convergence (CF1, CF2, CF3, CF4, CF5)

Figure 7: Conceptual illustration of the different choices for three variables

3.5 Data Analysis Methodology

The results of all simulations are automatically collected in a spreadsheet that shows the simulation progression, maximum observed value of the objective function, number of iterations for each run and its runtime. Having collected the simulation results, the data is further analysed to reveal any patterns, i.e. how different combinations of the chosen independent variables influence the optimisation results.

4 Experimental Results

The purpose of experimental data collection and analysis is twofold. Firstly, to determine whether reservoir engineering expert knowledge plays a major role in the well placement optimisation through a provision of a good quality initial approximation. Secondly, to figure out whether adjusting optimiser parameters can reduce the need for prior expert knowledge.

4.1 Summary of Experimental Results

Each HOPSPACK simulation produces a custom-made run summary file that contains well locations and ultimate recovery for each evaluated set of optimisation variables. The data is processed to extract ultimate recovery value for each iteration. These values are filtered to only show an increase in ultimate recovery over progressing iterations. These ultimate recovery data series are then collected in one chart (see Figure 8) in order to identify any obvious trends.

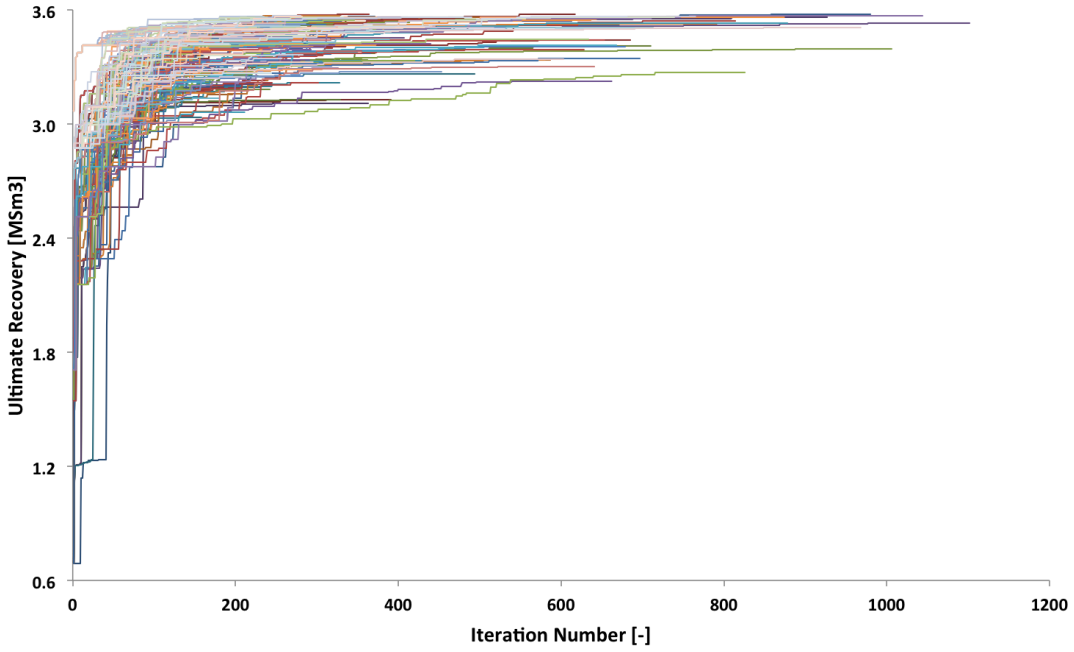


Figure 8: Ultimate recovery improvement over progressing iterations

The general trend observed for most simulations is that there is a rapid improvement at the beginning of each simulation, followed by a slow convergence to the final solution. The two important pieces of information that can be extracted from Figure 8 are the value of ultimate recovery and the number of iterations required to find that value. The figure shows that some HOPSPACK simulation scenarios are able to converge to a higher ultimate recovery than others. In addition, there is a large variation in the number of iterations HOPSPACK needed in order to converge to a solution.

Figure 8 makes it difficult to draw any conclusions due to the large amount of data. For this reason, the most important simulation results, namely the ultimate recovery and the number of iterations, are extracted and presented in Figure 9 and in Appendices B1-B7.

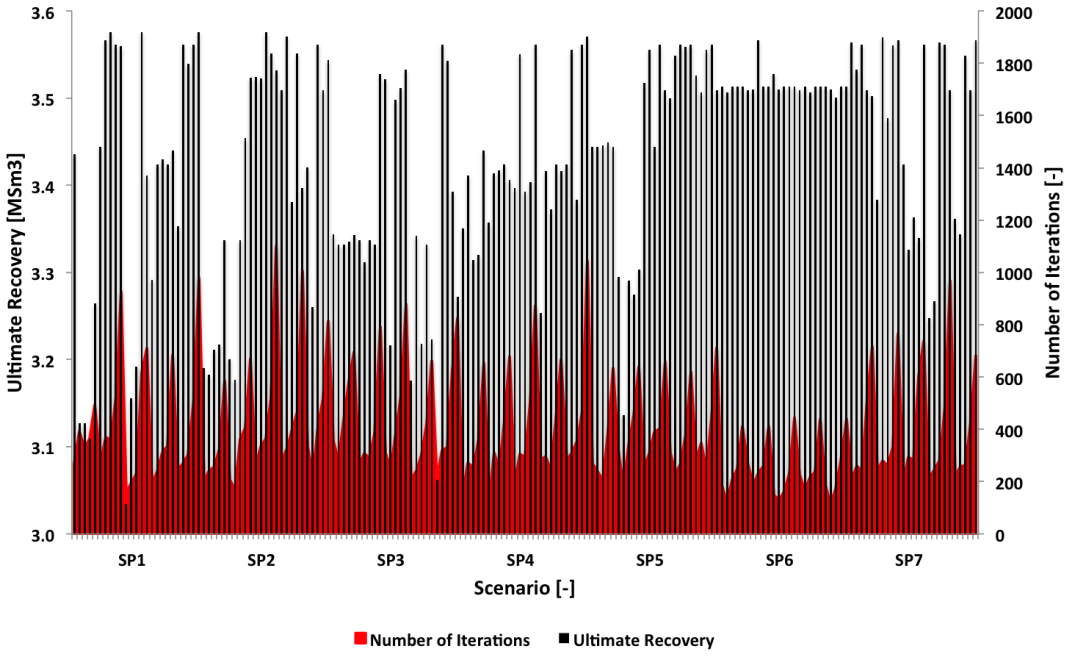


Figure 9: Ultimate recovery and number of iterations for each simulation run

Evidently, there is a large variation in ultimate recovery, ranging from 3.03 MSm^3 to 3.58 MSm^3. What is interesting is that scenarios with SP6 initial well placement, which is considered the most near-optimal, consistently produce high, but not the highest, ultimate recovery values as shown in Figure 9. This is unlike other scenarios with different initial well placements, which generally produce a wider range of ultimate recovery values.

There is also an obvious periodicity pattern in the number of iterations. This is a direct consequence of increasing both, the initial step size and the contraction factor, which ultimately affect the speed of convergence. Figure 10 demonstrates how increasing the contraction factor affects the number of distinct step sizes used by HOPSPACK in the search process. In this case, the largest initial step size (SS5) is used to illustrate the point. However, the same applies to all initial step sizes.

Evidently, there is a significant difference between the largest (CF5) and smallest (CF1) contraction factors. In the case of the largest contraction factor, there are 39 distinct step sizes, while there are only 7 of them in the case of the smallest contraction factor. Obviously, a choice of different contraction factors affects the speed of convergence. This explains why optimisation scenarios with CF5 take a lot longer to converge to the final solution than those with CF1.

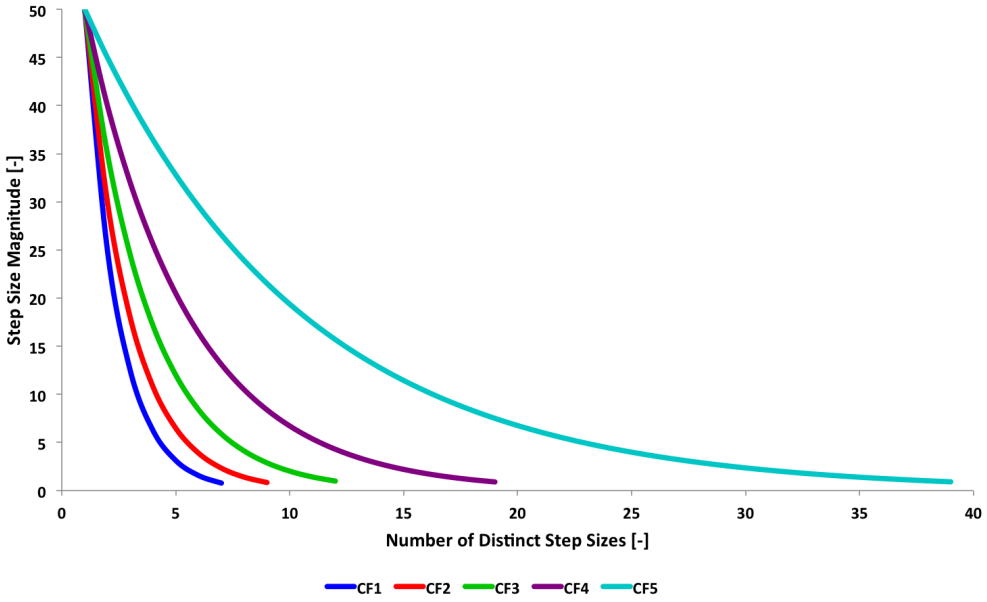


Figure 10: Step size (SP5) evolution with different contraction factors

4.2 Significance of Initial Well Placement

It is hypothesised that the quality of initial approximation would have a large impact on the result of optimisation. In order to understand whether the hypothesis is correct, all optimisation scenarios with the same initial well placement are averaged and presented in Figure 11.

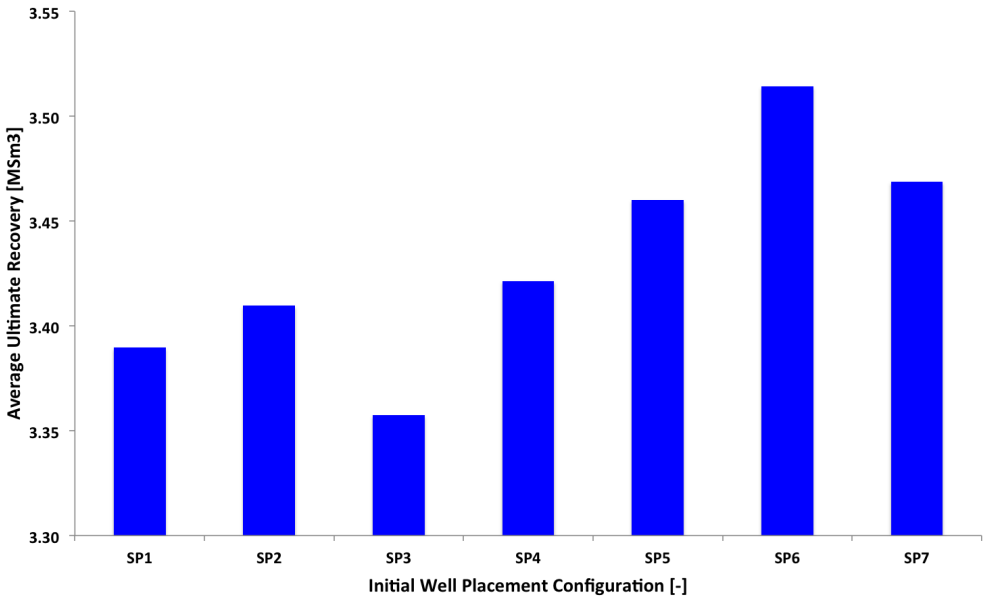


Figure 11: Influence of initial well placement on ultimate recovery

Figure 11 shows that scenarios with near-optimal initial well placement (SP5-SP7) produce, on average, better results than those with suboptimal initial well placement (SP1-SP4). The difference between the two groups is a direct consequence of having a good quality initial approximation. Similar difference is also evident in Figure 12, which shows the average ultimate recovery of each group of scenarios, namely SP1-SP4 and SP5-SP7. The difference between the two groups is roughly 3 %.

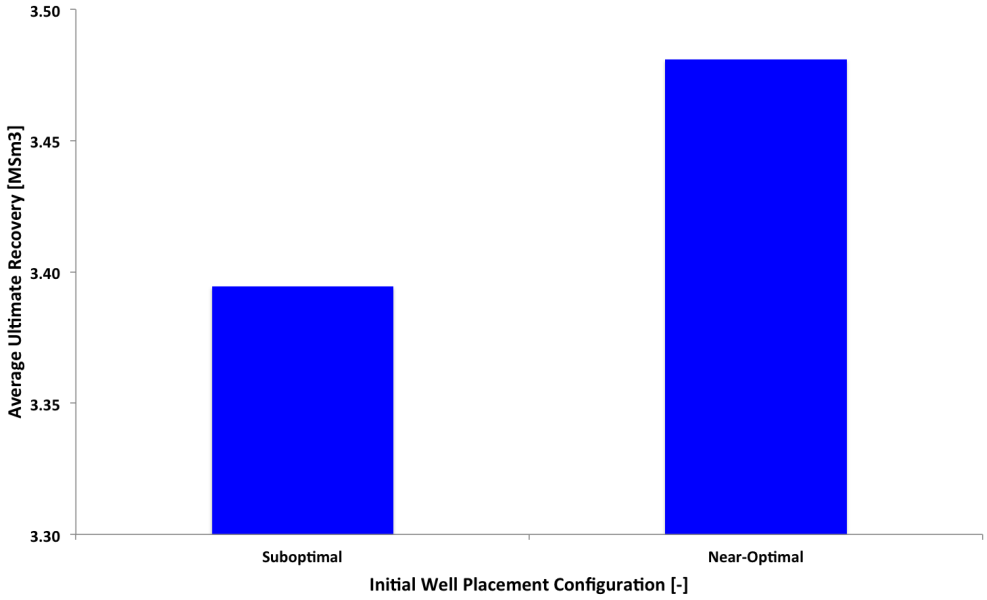


Figure 12: Comparison of different well placement strategies

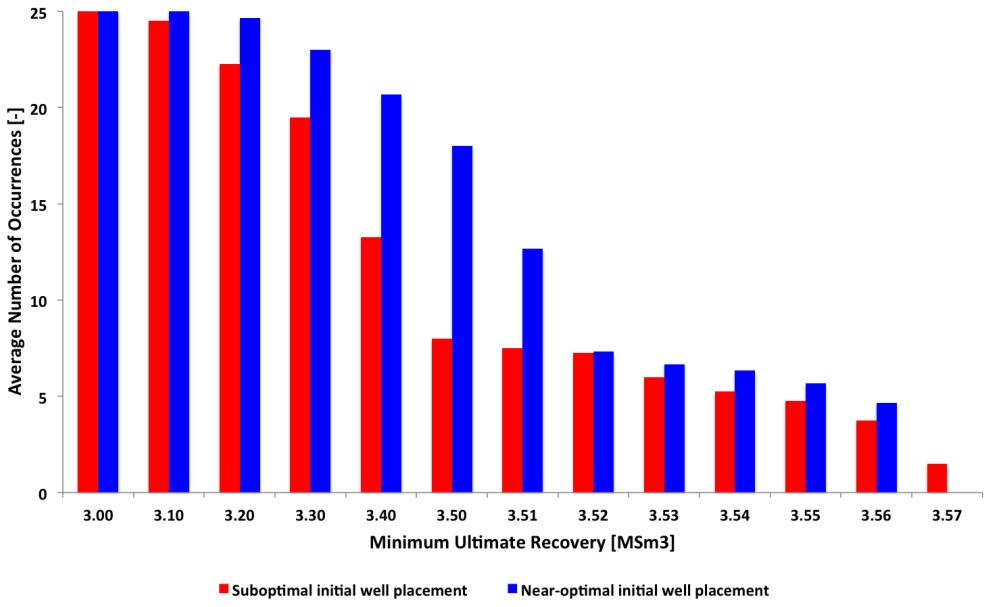


Figure 13: Occurrence count at particular threshold ultimate recovery

As mentioned earlier, scenarios with SP6 initial well placement configuration consistently produce high recoveries compared to other scenarios. This is reflected in Figure 11 by the fact that SP6 has the highest average ultimate recovery.

An alternative way to interpret the data is to count the average number of occurrences optimisation scenarios were able to surpass a certain minimum threshold value. Figure 13 presents this information for the aforementioned groups (SP1-SP4 and SP5-SP7). The figure shows that absolutely all scenarios converged to a solution that is equal to or greater than 3 MSm^3.

According to Figure 13, near-optimal initial well placement configurations outnumber the suboptimal ones for the majority of cases. Interestingly enough, the highest observed ultimate recovery (which is also presumed to be the global maximum) is achieved in a scenario with suboptimal initial well placement, which is likely an exception to the rule for more complex cases. The highest observed ultimate recovery is 3.58 MSm^3.

4.3 Significance of Optimiser Parameters

One of the objectives of this study is to understand whether altering optimiser parameters can significantly reduce the dependence on expert knowledge for providing a good initial approximation when solving the well placement optimisation problem. The results of numerical experiments suggest that improvements due to better initial well placement and a proper choice of optimiser parameters are of the same order of magnitude. On average, better ultimate recovery can be obtained using a better initial well placement configuration. But, the shortcomings of poor well placement can be compensated for by a correct choice of optimiser parameters.

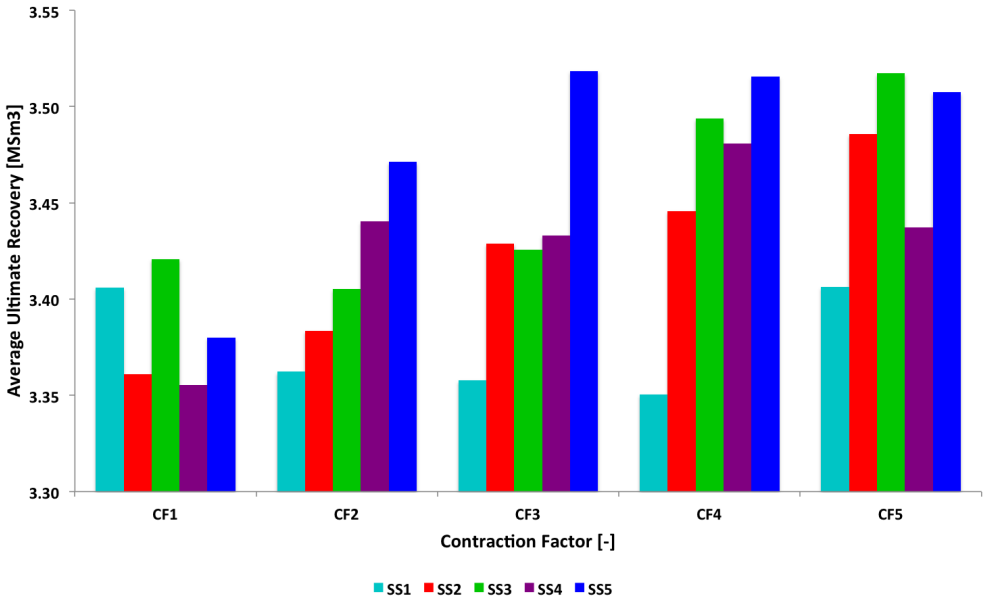


Figure 14: Influence of the optimisation parameters on ultimate recovery

Figure 14 shows how the two chosen optimiser parameters (initial step size and contraction factor) influence the average ultimate recovery. It follows from this figure that it is important to use an appropriate set of optimiser parameters, as they have an obvious impact on optimisation results. There is a general tendency for average ultimate recovery to increase with increasing initial step size and contraction factor values. The significance of each parameter is addressed in the following subsections.

4.3.1 Initial Step Size

As mentioned earlier, initial step size influences the explorative property of this particular optimisation algorithm. Provided that the grid size is 60×60 blocks, the initial steps sizes are chosen accordingly to simulate different explorative behaviours. The initial step sizes are 10 (SS1), 20 (SS2), 30 (SS3), 40 (SS4) and 50 (SS5) blocks. Using a larger initial step size means that the optimiser explores the objective function in a more global fashion.

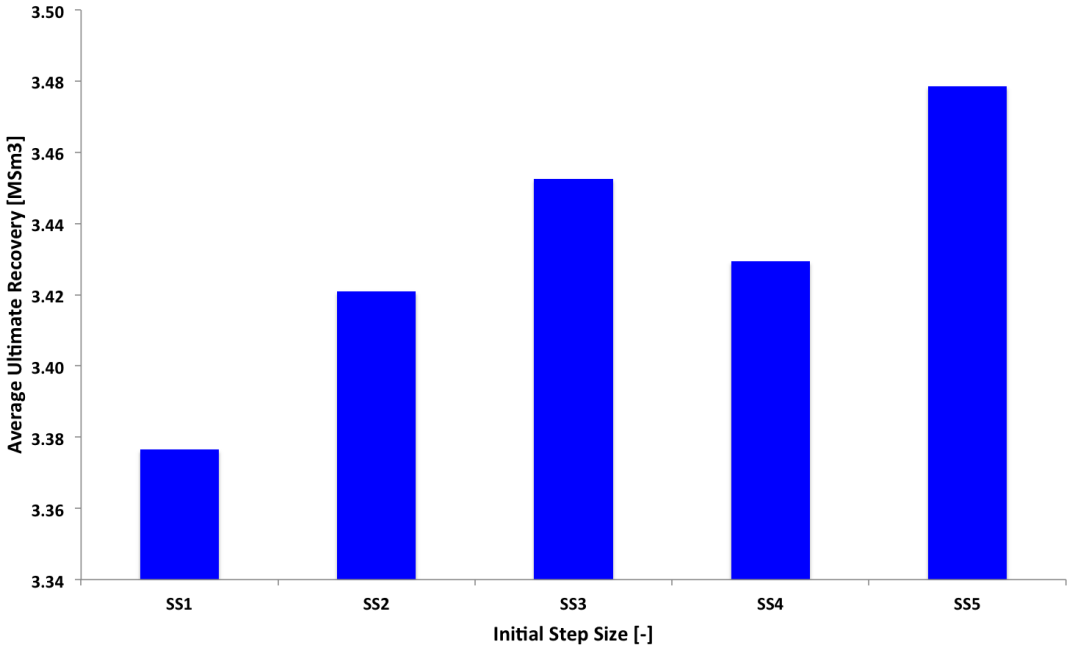


Figure 15: Influence of initial step size on average ultimate recovery

Figure 15 illustrates the way initial step size influences average ultimate recovery. Evidently, SS3 and SS5 initial step sizes yield the best results, while SS1 initial step size corresponds the worst average ultimate recovery. According to the HOPSPACK manual, it is good practice to use a large initial step size [16]. This advice is supported by the experimental results.

4.3.2 Contraction Factor

Contraction factor is a parameter that controls the rate of step size decrease. As a consequence, increasing the contraction factor decreases the speed of convergence due to the fact that it determines the number of distinct step sizes throughout the simulation (see Figure 10). Contraction factors chosen for this study are: 0.5 (CF1), 0.6 (CF2), 0.7 (CF3), 0.8 (CF4) and 0.9 (CF5). When the optimiser deems it appropriate, it multiplies the current stencil by the contraction factor to obtain a new stencil and thus narrow the search. The contraction factor has to be less than unity for HOPSPACK to be able to converge.

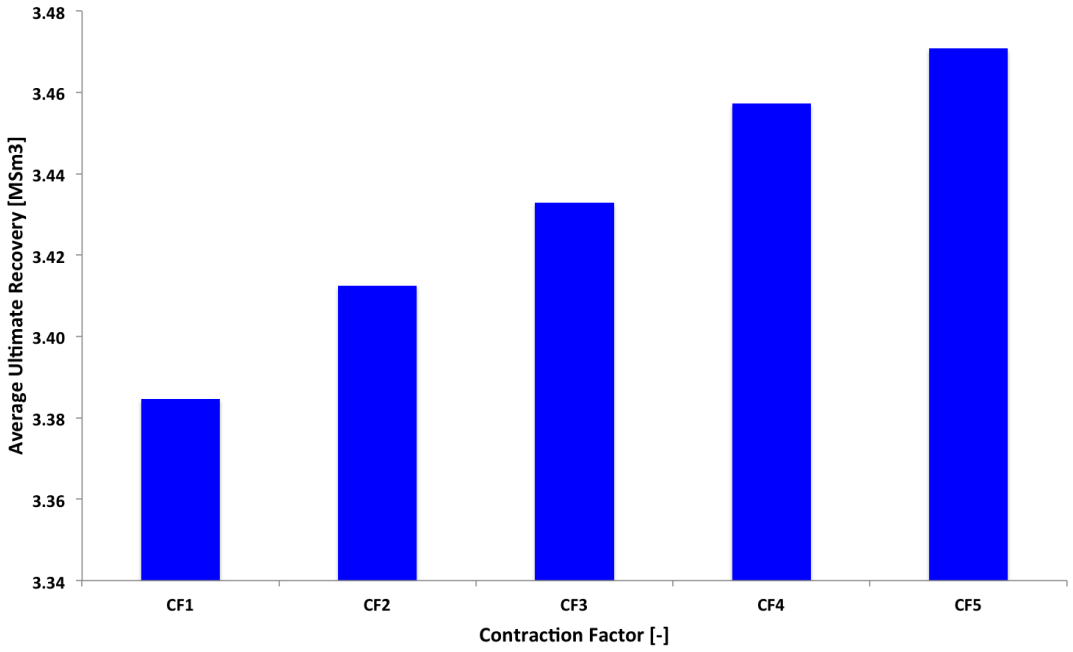


Figure 16: Influence of contraction factor on average ultimate recovery

As seen in Figure 16, there is a monotonous increase in the average ultimate recovery with an increasing contraction factor. It is clear from this figure that best optimisation results arise from using a large contraction factor (CF5 in this case). It follows that HOPSPACK is able to find better objective function values with more distinct step sizes, that are enabled by a large contraction factor.

4.4 Optimal Well Placement

Four out of 175 scenarios are able to converge to what appears to be the global maximum. The ultimate recovery for those scenarios is 3.58 MSm^3. The best well placement configuration found by HOPSPACK is presented in Figure 17 and Table 2. Figure 17 shows how HOPSPACK moved the wells to achieve optimality.

Figure 18 shows final oil saturation for the base-case (SP7) and best-case scenarios. Evidently, the major part of mobile oil is displaced by water in both cases. However, optimal well placement in the best-case scenario contributes to a better and more uniform volumetric sweep (see Figure 18b).

Table 2: Best-case scenario well coordinates

Well Name	**Well Coordinates**	
	X	**Y**
INJ1	30	28
PROD1	7	10
PROD2	6	50
PROD3	43	55
PROD4	55	9

Interestingly, well placement in the best-case scenario resembles an inverted five-spot pattern with an injector in the middle and four producers around it. However, in this case, well locations are adjusted to compensate for heterogeneous porosity and permeability fields. What is surprising is that HOPSPACK is able to derive this solution from a SP1 initial well placement, which is considered suboptimal.

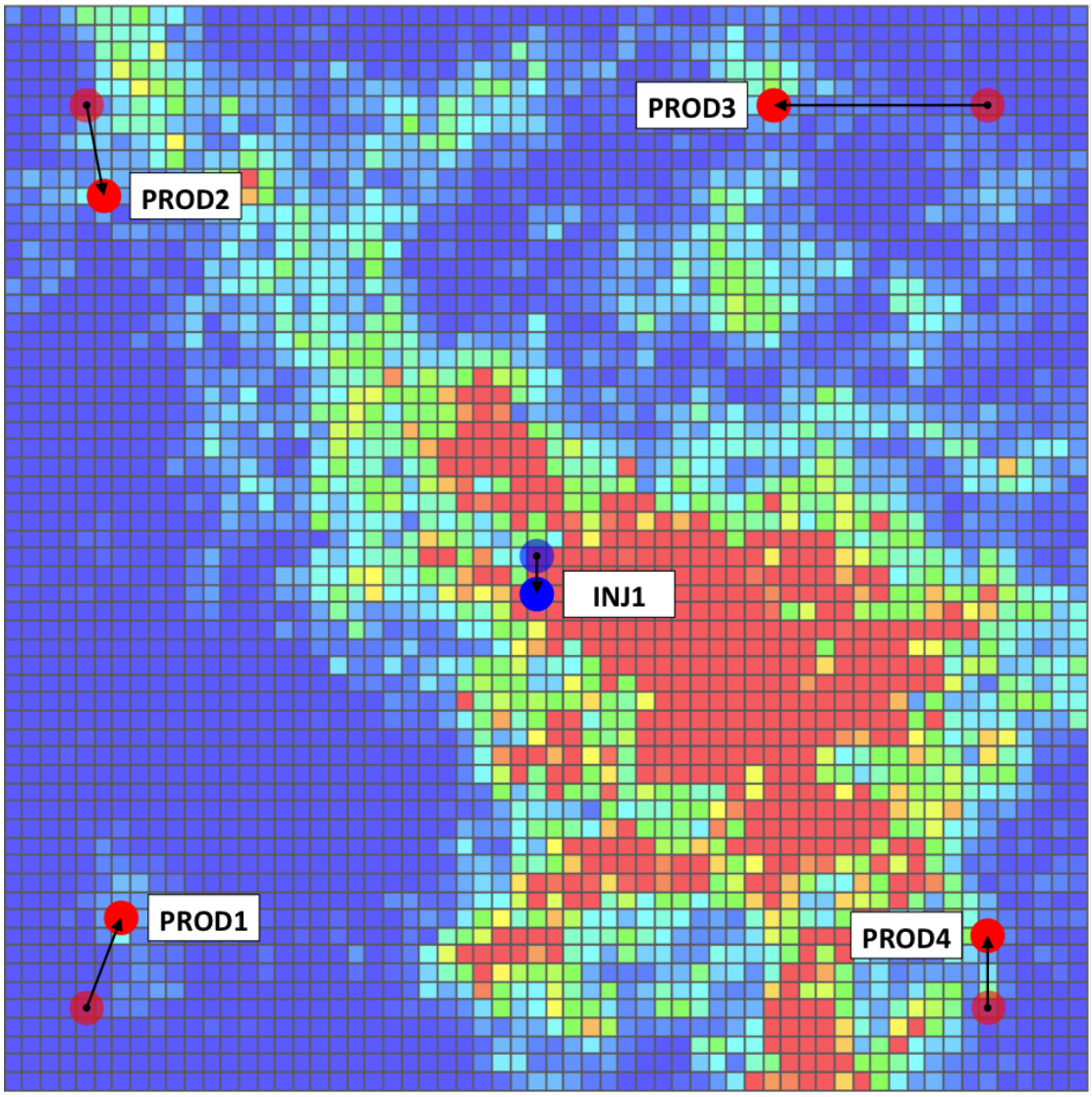
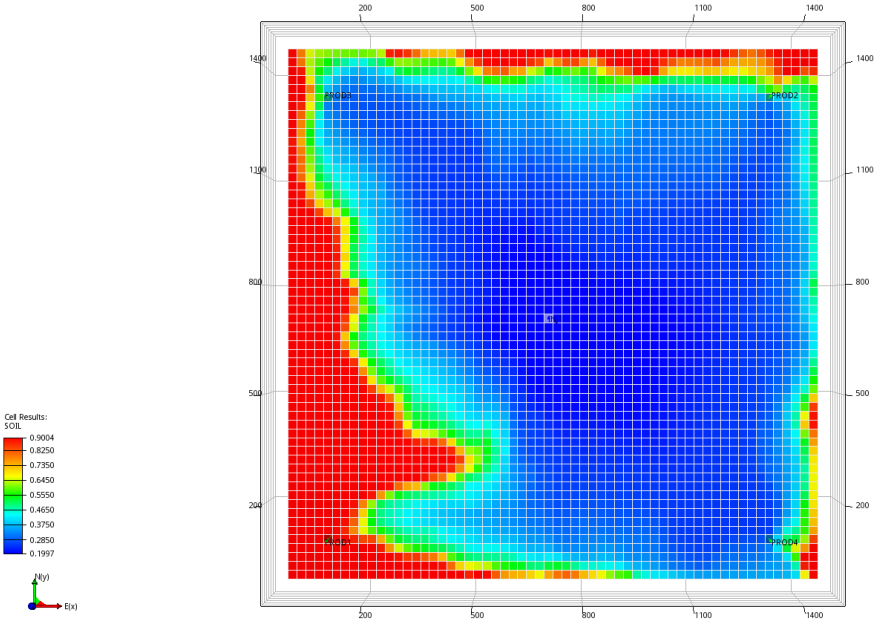
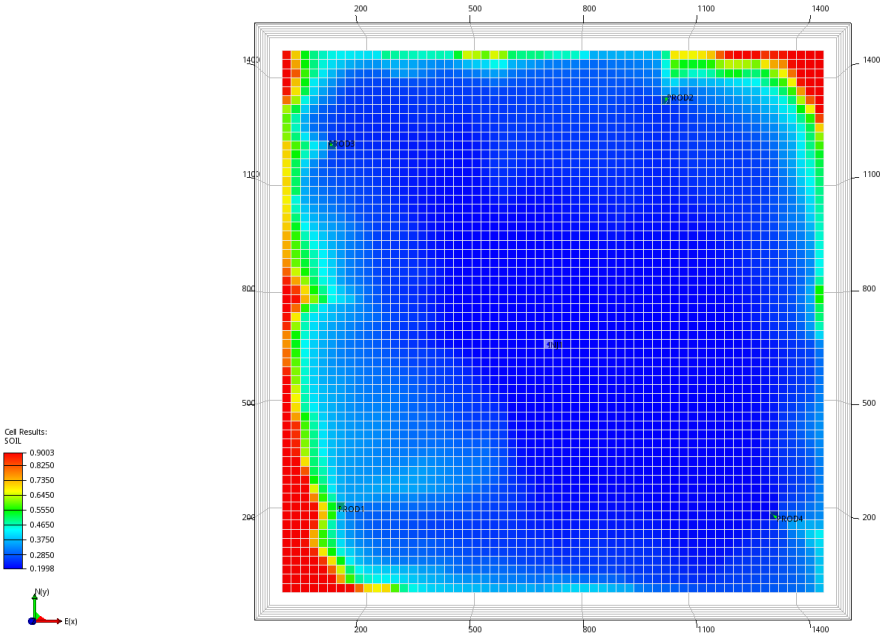


Figure 17: Initial and optimal well locations



(a) Final oil saturation of the base-case (SP7) scenario



(b) Final oil saturation of the best-case scenario

Figure 18: Initial and optimal final oil saturations

5 Discussion

5.1 Interpretation of Numerical Experiments

According to the hypothesis, reservoir engineering expert knowledge plays an important role in well placement optimisation through a provision of a good initial approximation. The results of numerical experiments show that this hypothesis holds true. There are two main reasons why the results of numerical experiments verify this hypothesis.

Firstly, Figure 11 makes it clear that the optimisation results are significantly improved as a result of using near-optimal initial well placement. HOPSPACK proves to be able to find good solutions for scenarios with near-optimal well placement. This result is good and somewhat expected, because HOPSPACK is known to employ a local search method. Therefore, it is expected to more likely find a local optimum in the vicinity of the initial approximation.

Secondly, scenarios with the most near-optimal initial well placement configuration (SP6) consistently produce high ultimate recovery values. Unexpectedly, the absolute best solution (apparent global optimum) comes from the suboptimal initial well placement, which is likely an exception to the rule for more complex cases. In fact, HOPSPACK converged to this solution on several occasions. HOPSPACK is somehow able to avoid the intentional trap (poor initial approximation) and rearrange the wells into a more favourable configuration.

For the aforementioned reasons, reservoir engineering expert knowledge that materialises in a better initial approximation for well placement plays an important role in well placement optimisation for this particular simplistic, two-dimensional reservoir model with five wells.

Figure 11 and Figure 12 may help reinforce this statement. It was expected to see that a better initial approximation would lead to a significantly better optimisation result. Indeed, scenarios that take advantage of better initial approximation outperform those with suboptimal initial well placement.

This obvious difference may be partly attributed to the stencil size when HOPSPACK is in the vicinity of a near-optimal solution. A HOPSPACK simulation that starts with a good initial approximation has more time to explore the objective function around a known near-optimal solution than a simulation that starts with a poor initial approximation. Therefore, by the time a simulation with unfavourable initial conditions reaches the near-optimal solution, its stencil size is significantly smaller than that of a simulation that starts in the vicinity of a near-optimal solution. This difference in temporal stencil size leads to different degrees of exploration in the vicinity of a near-optimal solution, which causes the difference in ultimate recovery.

The choice of a good quality initial approximation appears to have the same impact on the results of optimisation as an adequate choice of optimiser parameters. Results suggest that improvements due to each of the two factors are of the same order of magnitude.

It is hypothesised that an appropriately configured optimisation algorithm may significantly reduce the need for prior expert knowledge and near-optimal initial

approximation when well placement problems are being solved. Results of the numerical experiments with the two-dimensional model presented in Section 3 clearly confirm this hypothesis. Figure 15 and Figure 16 show a near-systematic correlation between both optimiser parameters and better oil recovery results.

According to the results of numerical experiments, there is a clear correlation between the contraction factor and average ultimate recovery. An increase in ultimate recovery is a direct consequence of the use of a larger contraction factor. A larger contraction factor prolongs the search by allowing the stencil to shrink at a lower rate. This, in turn, enables HOPSPACK to explore a larger portion of the search space. Of course, this comes at an expense of computation time. In fact, the red peaks (longest computation time) that appear in Figure 9 correspond to scenarios with the largest contraction factor (CF5).

The results also indicate that there is a correlation between the initial step size and average ultimate recovery, yet not as pronounced as that of the contraction factor. It is clear from Figure 15 that two particular choices of initial step size stand out, namely SS3 (30 grid blocks) and SS5 (50 grid blocks). Evidently, the latter choice of the initial step size leads to a higher average ultimate recovery than the former one. There may potentially be several reasons why the two choices of the initial step size stand out.

Presumably, SS3 stands out because it represents a balance between local and global exploration of the search space. The stencil is exactly half the size of the grid. Thereby, HOPSPACK does not make drastic jumps across the whole grid as it may do in the case of SS5. It seems that using too large of a step size may actually have a negative impact on the optimisation process because the stencil becomes comparable to the size of the grid. This negative impact may be particularly pronounced for the cases that start with a near-optimal initial well placement. For these cases, a large step size may divert the optimiser away from a nearby optimum.

Scenarios with SS5 produced the highest average ultimate recovery. SS5 is the largest step size used in the numerical experiments and is supposed to imitate a global search because the stencil is comparable to the size of the grid. A large step size allows HOPSPACK to move the wells around with greater freedom. For this reason, HOPSPACK is able to find good solutions to scenarios with poor initial well placement. A large initial step size somewhat delays the convergence to a local optimum, thus allowing HOPSPACK to conduct a more thorough exploration of the search space. According to HOPSPACK user manual, it is generally better to use a large initial step size [16]. This statement is directly confirmed and reinforced by the results of numerical experiments.

All things considered, the role of reservoir engineering expert knowledge in the process of well placement optimisation for this particular model appears to be important. However, the shortage of such knowledge mimicked by poor initial approximation could be compensated for by a proper choice of the optimiser parameters.

It is best to use a balanced initial step size to avoid diverting the optimiser away from a nearby optimum. This will ensure to maintain a balance between local and

global exploration. Otherwise, in the absence of a good initial approximation, it is best to use the largest initial step size to ensure global exploration. Finally, it is recommended to use the largest contraction factor to allow HOPSPACK find the best solution to the well placement problem. When it comes to the choice of the contraction factor, it is important to consider the trade-off between the ultimate recovery and computation time.

5.2 Guiding Principles for Optimal Well Placement

Optimal well placement is concerned with maximising volumetric sweep efficiency and minimising fluid energy loss along with minimising well interference [17]. It may be helpful to consider two reservoirs, namely homogeneous and heterogeneous, in order to derive the guiding principles for optimal well placement.

First, let us consider the square two-dimensional model with four producers and one injector (like the one used in this thesis), but assume a homogeneous distribution of reservoir properties. Anyone with a basic understanding of reservoir engineering would find it very intuitive to place the producers in each corner of the grid, at an equal distance away from the injector, which would be located exactly at the centre of the grid. There are two main reasons why the wells would be arranged in a so-called inverted five-spot pattern.

Firstly, such well placement may be attributed to the movement of water front through the reservoir. In a homogeneous reservoir, the front would spread symmetrically away from the injector towards each producer, considering that the producers operate at same regime. Consequently, the producers are placed equidistantly from the injector to equalise water front travel time. Such equidistant well placement along with a unified well control strategy helps maintain the symmetric shape of the water front and avoid earlier water breakthrough in one well. The symmetry also helps minimise interference between wells.

Secondly, placing producers in each corner of the grid maximises water front travel time from injector to each producer. This, in turn, maximises the volumetric sweep efficiency.

Similar guiding principles may apply to the same reservoir model, but with a heterogeneous distribution of reservoir properties. The wells need to be placed in a way that would allow the water front to spread more uniformly in all directions. This would also equalise water front travel time from injector to each production well. Just like in the homogeneous case, the wells need to be located far enough from the injector to maximise sweep efficiency and far apart from each other to reduce their interference.

In general, lateral heterogeneity in reservoir properties leads to a loss of symmetry. The travel time of the water front from injector to each producer placed at equal distances is no longer the same due to variations in horizontal permeability. In order to compensate for this difference and equalise water front travel times, the wells would have to be moved from their initial symmetric (and now suboptimal) locations to form an irregular (skewed) inverted five-spot pattern as shown in Figure 17.

The results of the numerical experiments make it possible to derive some guiding principles behind optimal well placement in such a two-dimensional reservoir with a fixed number of wells. These guiding principles may lay a foundation for writing an application to reservoir optimisers that would mimic application of expert knowledge through a provision of a good initial approximation.

5.3 Future Work

It is important to appreciate that numerical experiments in this thesis are conducted on a two-dimensional reservoir model. Surely, the optimisation process is somewhat simpler that in a three-dimensional case, because each well has only one connection and vertical heterogeneity (if any) is not important in a two-dimensional case. In the case of three-dimensional reservoir models, however, the optimisation process will not be as straightforward. It will require more complex rules (and constraints) to guide well placement. An additional optimisation parameter will be the number of grid connections per well. From a mathematical standpoint, an additional degree of flexibility (due to an extra dimension) and a significant increase in the number of optimisation parameters will make the optimisation problem a lot more challenging.

There is no doubt that an expertise of a skilled reservoir engineer will be more essential for a three-dimensional problem, just because of the fact that optimal well shape no longer has to be vertical. More complex well geometry can be considered, such as deviated (inclined), horizontal and multilateral wells. Thus, the process of optimisation can most probably be simplified in case if a good initial approximation for well locations and shapes is given at start.

The work done in this thesis may be extended to a three-dimensional case with the same number and type of wells, but with vertical heterogeneity in addition. The wells would have to be vertical, at least in the beginning, to keep the optimisation problem complexity under control. In addition to well locations, it is possible to try to optimise the number of connections per well and their positions along the reservoir section of the wells. This would reveal whether a chosen optimisation algorithm may be used for solving more complicated reservoir engineering optimisation problems more typical for real life situations.

6 Conclusion

There is generally a sceptical attitude in the petroleum industry towards the use of optimisation techniques for solving field development problems. Nevertheless, an increasing availability of computing power and a strong motivation to make optimal field development decisions have been inspiring researchers to apply derivative-free optimisation methods to field development problems in the recent years. This work builds on the recent research. It is conducted in an attempt to help dispel some of that scepticism and to inspire additional research in this field.

This work describes an effort to better understand the role of reservoir engineering expert knowledge in well placement optimisation at the stage of field development planning and figure out whether an appropriate choice of optimiser parameters can help reduce the dependence on experience-based knowledge.

For the purpose of conducting numerical experiments, it was decided to use the HOPSPACK optimisation software package for its novelty, advanced nature and computational efficiency. In order to integrate the ECLIPSE reservoir simulator with HOPSPACK, a purpose-specific optimiser-simulator interface was developed as a part of this project.

The interpretation of results of the numerical experiments has led to two major conclusions for the particular reservoir model used:

- Reservoir engineering expert knowledge plays an important role in the process of well placement optimisation by providing a good quality initial approximation;
- A proper choice of the optimiser parameters that allows for a wide exploration of the search space may significantly reduce the dependence on such expert knowledge.

One particular implication of this study is that engineers should have a thorough understanding of the optimisation algorithm they use to solve a problem. Having such understanding, they can significantly enhance the well placement optimisation workflow.

The numerical optimisation experiments were conducted on a two-dimensional reservoir model, which in many cases is an oversimplification of the typical real life situations. As a direct consequence of such choice of model, the study encountered two obvious limitations.

Firstly, the conclusions reached in this work may not necessarily apply to more complex, three-dimensional reservoir models, which better represent typical real life situations. Therefore, the study should be extended to more realistic three-dimensional cases.

Secondly, numerical experiments only considered a vertical well shape. In more realistic reservoir models, the optimal well shape no longer has to be vertical, which is why subsequent research work may address more complex well geometry.

This study compliments the previous research efforts by others and helps prove that modern advanced optimisation techniques deliver promising results for solving field development problems.

Undoubtedly, not a single important decision in the petroleum industry is being made without consulting an expert. However, it is very likely that the experts will be taking a greater advantage of optimisation techniques for the decision-making process in a not so distant future.

Bibliography

[1] B. Guyaguler and R. N. Horne, "Uncertainty assessment of well placement optimization," *SPE Journal*, vol. 7, 2001.

[2] O. J. Isebor, D. E. Ciaurri, and L. J. Durlofsky, "Generalized field-development optimization with derivative-free procedures," *SPE Journal*, vol. 19, 2014.

[3] J. E. Hicken and J. Alsonso, "Gradient-free optimization." Handout.

[4] M. C. Bellout, D. E. Ciaurri, L. J. Durlofsky, B. Foss, and J. Kleppe, "Joint optimization of oil well placement and controls," *Computational Geosciences*, vol. 16, 2012.

[5] O. Kramer, D. E. Ciaurri, and S. Koziel, "Derivative-free optimisation," *Computational Optimization, Methods and Algorithms*, vol. 356, 2011.

[6] O. J. Isebor, "Constrained production optimization with an emphasis on derivative-free methods," Master's thesis, Stanford University, 2009.

[7] M. Jesmani, M. C. Bellout, R. Hanea, and B. Foss, "Particle swarm optimization algorithm for optimum well placement subject to realistic field development constraints," *SPE Journal*, 2015.

[8] J. E. Onwunalu, *Optimisation of Field Development Using Particle Swarm Optimisation and New Well Pattern Descriptions*. PhD thesis, Stanford University, 2010.

[9] P. D. Hough, T. G. Kolda, and V. J. Torczon, "Asynchronous parallel pattern search for nonlinear optimization," *SIAM Journal on Scientific Computing*, vol. 23, 2001.

[10] D. E. Ciaurri, T. Mukerji, and L. Durlofsky, "Derivative-free optimization for oil field operations," *Computational Optimization and Applications in Engineering and Industry*, vol. 359, 2011.

[11] T. G. Kolda and V. J. Torczon, "Understanding asynchronous parallel pattern search," *High Performance Algorithms and Software for Nonlinear Optimization*, vol. 82, 2003.

[12] G. A. Gray and T. G. Kolda, "Algorithm 856: Appspack 4.0: Asynchronous parallel pattern search for derivative-free optimization," *ACM Transactions on Mathematical Software*, vol. 32, 2006.

[13] K. Y. Lee and J.-B. Park, "Application of particle swarm optimization to economic dispatch problem: Advantages and disadvantages," *IEEE*, 2006.

[14] Wikipedia, "Particle swarm optimization," 2015.

[15] M. C. . M. Blunt, "Tenth spe comparative solution project: A comparison of upscaling techniques," *SPE Journal*, 2001.

[16] T. D. Plantenga, "Hopspack 2.0 user manual," Tech. Rep. SAND2009-6265, Sandia National Laboratories, Albuquerque, NM and Livermore, CA, October 2009.

[17] M. Shepherd, *Oil field production geology: AAPG Memoir 91*, ch. Well Patterns, pp. 239–240. AAPG, 2009.

Appendix A - HOPSPACK Configuration File

Figure 19: HOPSPACK configuration file

```
# ************************************************************************************************
#                        HOPSPACK: Hybrid Optimization Parallel Search Package
#                              Copyright 2009-2010 Sandia Corporation
#
#   Under the terms of Contract DE-AC04-94AL85000 with Sandia Corporation,
#   the U.S. Government retains certain rights in this software.
# ************************************************************************************************

# HOPSPACK - Configuration SPX_SSY_CFZ

#-- PROBLEM DEFINITION SUBLIST ----------------------------
#     Parameter Name              Type          Value                                  Comment
@ "Problem Definition"
      "Number Unknowns"           int           10
      "Lower Bounds"              vector 10     5 5 5 5 5 5 5 5 5 5
      "Upper Bounds"              vector 10     55 55 55 55 55 55 55 55 55 55
      "Scaling"                   vector 10     1 1 1 1 1 1 1 1 1 1
      "Variable Types"            charvec       O O O O O O O O O O
      "Objective Type"            string        "Maximize"
#     "Initial X"                 vector 10     5  5  55 5  55 55 5  55 30 30          # SP1
#     "Initial X"                 vector 10     5  55 5  5  55 5  55 55 30 30          # SP2
#     "Initial X"                 vector 10     55 55 5  55 5  5  55 5  30 30          # SP3
#     "Initial X"                 vector 10     55 5  55 55 5  55 5  5  30 30          # SP4
#     "Initial X"                 vector 10     30 30 15 15 45 15 45 45 15 45          # SP5
#     "Initial X"                 vector 10     30 30 10 10 50 10 50 50 10 50          # SP6
#     "Initial X"                 vector 10     30 30 5  5  55 5  55 55 5  55          # SP7
      "Display"                   int           2
@@
#------------------------------------------------------------

#-- EVALUATOR SUBLIST -------------------------------------
#     Parameter Name              Type          Value                                  Comment
@ "Evaluator"
      "Evaluator Type"            string        "System Call"
      "Executable Name"           string        "CaseRunnerHOPSPACK"
      "Input Prefix"              string        "in"
      "Output Prefix"             string        "out"
      "Debug Eval Worker"         bool          false
      "Save IO Files"             bool          false
@@
#------------------------------------------------------------

#-- EVALUATOR SUBLIST -------------------------------------
#     Parameter Name              Type          Value                                  Comment
@ "Mediator"
      "Citizen Count"             int           1
      "Number Processors"         int           2
      "Number Threads"            int           4
      "Maximum Evaluations"       int           2000
      "Solution File"             string        "solution"
      "Display"                   int           3
      "Synchronous Evaluaions"    bool          false
@@
#------------------------------------------------------------

#-- EVALUATOR SUBLIST -------------------------------------
#     Parameter Name              Type          Value                                  Comment
@ "Citizen 1"
      "Type"                      string        "GSS"
      "Step Tolerance"            double        1.0
#     "Contraction Factor"        double        0.5                                    # CF1
#     "Contraction Factor"        double        0.6                                    # CF2
#     "Contraction Factor"        double        0.7                                    # CF3
#     "Contraction Factor"        double        0.8                                    # CF4
#     "Contraction Factor"        double        0.9                                    # CF5
      "Epsilon Max"               double        2.0
#     "Initial Step"              double        10.0                                   # SS1
#     "Initial Step"              double        20.0                                   # SS2
#     "Initial Step"              double        30.0                                   # SS3
#     "Initial Step"              double        40.0                                   # SS4
#     "Initial Step"              double        50.0                                   # SS5
      "Display"                   int           1
@@
#------------------------------------------------------------
```

Appendix B1 - SP1 Configuration Parameters

Figure 20: HOPSPACK configuration parameters for SP1 scenarios

Scenario	Initial Well Placement					Initial Step Size	Contraction Factor	Ultimate Recovery	Number of Iterations
[-]	[INJ1(X,Y)]	[PROD1(X,Y)]	[PROD2(X,Y)]	[PROD3(X,Y)]	[PROD4(X,Y)]	[grid cells]	[-]	MSm3	[-]
SP1_SS1_CF1	(5,5)	(55,5)	(55,55)	(5,55)	(30,30)	10	0,5	3,44	268
SP1_SS1_CF2	(5,5)	(55,5)	(55,55)	(5,55)	(30,30)	10	0,6	3,13	391
SP1_SS1_CF3	(5,5)	(55,5)	(55,55)	(5,55)	(30,30)	10	0,7	3,13	329
SP1_SS1_CF4	(5,5)	(55,5)	(55,55)	(5,55)	(30,30)	10	0,8	3,11	363
SP1_SS1_CF5	(5,5)	(55,5)	(55,55)	(5,55)	(30,30)	10	0,9	3,26	494
SP1_SS2_CF1	(5,5)	(55,5)	(55,55)	(5,55)	(30,30)	20	0,5	3,44	275
SP1_SS2_CF2	(5,5)	(55,5)	(55,55)	(5,55)	(30,30)	20	0,6	3,57	371
SP1_SS2_CF3	(5,5)	(55,5)	(55,55)	(5,55)	(30,30)	20	0,7	3,58	358
SP1_SS2_CF4	(5,5)	(55,5)	(55,55)	(5,55)	(30,30)	20	0,8	3,56	528
SP1_SS2_CF5	(5,5)	(55,5)	(55,55)	(5,55)	(30,30)	20	0,9	3,56	927
SP1_SS3_CF1	(5,5)	(55,5)	(55,55)	(5,55)	(30,30)	30	0,5	3,03	158
SP1_SS3_CF2	(5,5)	(55,5)	(55,55)	(5,55)	(30,30)	30	0,6	3,16	196
SP1_SS3_CF3	(5,5)	(55,5)	(55,55)	(5,55)	(30,30)	30	0,7	3,19	231
SP1_SS3_CF4	(5,5)	(55,5)	(55,55)	(5,55)	(30,30)	30	0,8	3,58	617
SP1_SS3_CF5	(5,5)	(55,5)	(55,55)	(5,55)	(30,30)	30	0,9	3,41	710
SP1_SS4_CF1	(5,5)	(55,5)	(55,55)	(5,55)	(30,30)	40	0,5	3,29	193
SP1_SS4_CF2	(5,5)	(55,5)	(55,55)	(5,55)	(30,30)	40	0,6	3,42	239
SP1_SS4_CF3	(5,5)	(55,5)	(55,55)	(5,55)	(30,30)	40	0,7	3,43	312
SP1_SS4_CF4	(5,5)	(55,5)	(55,55)	(5,55)	(30,30)	40	0,8	3,42	337
SP1_SS4_CF5	(5,5)	(55,5)	(55,55)	(5,55)	(30,30)	40	0,9	3,44	685
SP1_SS5_CF1	(5,5)	(55,5)	(55,55)	(5,55)	(30,30)	50	0,5	3,35	246
SP1_SS5_CF2	(5,5)	(55,5)	(55,55)	(5,55)	(30,30)	50	0,6	3,56	270
SP1_SS5_CF3	(5,5)	(55,5)	(55,55)	(5,55)	(30,30)	50	0,7	3,54	303
SP1_SS5_CF4	(5,5)	(55,5)	(55,55)	(5,55)	(30,30)	50	0,8	3,56	519
SP1_SS5_CF5	(5,5)	(55,5)	(55,55)	(5,55)	(30,30)	50	0,9	3,58	980

Appendix B2 - SP2 Configuration Parameters

Figure 21: HOPSPACK configuration parameters for SP2 scenarios

Scenario	Initial Well Placement					Initial Step Size	Contraction Factor	Ultimate Recovery	Number of Iterations
[-]	[INJ1(X,Y)]	[PROD1(X,Y)]	[PROD2(X,Y)]	[PROD3(X,Y)]	[PROD4(X,Y)]	[grid cells]	[-]	MSm3	[-]
SP2_SS1_CF1	(5,55)	(5,5)	(55,5)	(55,55)	(30,30)	10	0,5	3,19	205
SP2_SS1_CF2	(5,55)	(5,5)	(55,5)	(55,55)	(30,30)	10	0,6	3,18	242
SP2_SS1_CF3	(5,55)	(5,5)	(55,5)	(55,55)	(30,30)	10	0,7	3,21	258
SP2_SS1_CF4	(5,55)	(5,5)	(55,5)	(55,55)	(30,30)	10	0,8	3,22	328
SP2_SS1_CF5	(5,55)	(5,5)	(55,5)	(55,55)	(30,30)	10	0,9	3,34	587
SP2_SS2_CF1	(5,55)	(5,5)	(55,5)	(55,55)	(30,30)	20	0,5	3,20	206
SP2_SS2_CF2	(5,55)	(5,5)	(55,5)	(55,55)	(30,30)	20	0,6	3,18	170
SP2_SS2_CF3	(5,55)	(5,5)	(55,5)	(55,55)	(30,30)	20	0,7	3,34	362
SP2_SS2_CF4	(5,55)	(5,5)	(55,5)	(55,55)	(30,30)	20	0,8	3,45	407
SP2_SS2_CF5	(5,55)	(5,5)	(55,5)	(55,55)	(30,30)	20	0,9	3,52	671
SP2_SS3_CF1	(5,55)	(5,5)	(55,5)	(55,55)	(30,30)	30	0,5	3,52	256
SP2_SS3_CF2	(5,55)	(5,5)	(55,5)	(55,55)	(30,30)	30	0,6	3,52	334
SP2_SS3_CF3	(5,55)	(5,5)	(55,5)	(55,55)	(30,30)	30	0,7	3,58	364
SP2_SS3_CF4	(5,55)	(5,5)	(55,5)	(55,55)	(30,30)	30	0,8	3,55	522
SP2_SS3_CF5	(5,55)	(5,5)	(55,5)	(55,55)	(30,30)	30	0,9	3,53	1102
SP2_SS4_CF1	(5,55)	(5,5)	(55,5)	(55,55)	(30,30)	40	0,5	3,51	278
SP2_SS4_CF2	(5,55)	(5,5)	(55,5)	(55,55)	(30,30)	40	0,6	3,57	328
SP2_SS4_CF3	(5,55)	(5,5)	(55,5)	(55,55)	(30,30)	40	0,7	3,38	401
SP2_SS4_CF4	(5,55)	(5,5)	(55,5)	(55,55)	(30,30)	40	0,8	3,55	469
SP2_SS4_CF5	(5,55)	(5,5)	(55,5)	(55,55)	(30,30)	40	0,9	3,40	1006
SP2_SS5_CF1	(5,55)	(5,5)	(55,5)	(55,55)	(30,30)	50	0,5	3,42	335
SP2_SS5_CF2	(5,55)	(5,5)	(55,5)	(55,55)	(30,30)	50	0,6	3,26	220
SP2_SS5_CF3	(5,55)	(5,5)	(55,5)	(55,55)	(30,30)	50	0,7	3,56	412
SP2_SS5_CF4	(5,55)	(5,5)	(55,5)	(55,55)	(30,30)	50	0,8	3,51	519
SP2_SS5_CF5	(5,55)	(5,5)	(55,5)	(55,55)	(30,30)	50	0,9	3,54	814

Appendix B3 - SP3 Configuration Parameters

Figure 22: HOPSPACK configuration parameters for SP3 scenarios

Scenario	Initial Well Placement					Initial Step Size	Contraction Factor	Ultimate Recovery	Number of Iterations
[-]	[INJ1(X,Y)]	[PROD1(X,Y)]	[PROD2(X,Y)]	[PROD3(X,Y)]	[PROD4(X,Y)]	[grid cells]	[-]	MSm3	[-]
SP3_SS1_CF1	(55,55)	(5,55)	(5,5)	(55,5)	(30,30)	10	0,5	3,34	355
SP3_SS1_CF2	(55,55)	(5,55)	(5,5)	(55,5)	(30,30)	10	0,6	3,33	279
SP3_SS1_CF3	(55,55)	(5,55)	(5,5)	(55,5)	(30,30)	10	0,7	3,33	436
SP3_SS1_CF4	(55,55)	(5,55)	(5,5)	(55,5)	(30,30)	10	0,8	3,34	580
SP3_SS1_CF5	(55,55)	(5,55)	(5,5)	(55,5)	(30,30)	10	0,9	3,34	697
SP3_SS2_CF1	(55,55)	(5,55)	(5,5)	(55,5)	(30,30)	20	0,5	3,34	262
SP3_SS2_CF2	(55,55)	(5,55)	(5,5)	(55,5)	(30,30)	20	0,6	3,31	309
SP3_SS2_CF3	(55,55)	(5,55)	(5,5)	(55,5)	(30,30)	20	0,7	3,34	291
SP3_SS2_CF4	(55,55)	(5,55)	(5,5)	(55,5)	(30,30)	20	0,8	3,33	385
SP3_SS2_CF5	(55,55)	(5,55)	(5,5)	(55,5)	(30,30)	20	0,9	3,53	791
SP3_SS3_CF1	(55,55)	(5,55)	(5,5)	(55,5)	(30,30)	30	0,5	3,52	311
SP3_SS3_CF2	(55,55)	(5,55)	(5,5)	(55,5)	(30,30)	30	0,6	3,22	245
SP3_SS3_CF3	(55,55)	(5,55)	(5,5)	(55,5)	(30,30)	30	0,7	3,50	387
SP3_SS3_CF4	(55,55)	(5,55)	(5,5)	(55,5)	(30,30)	30	0,8	3,51	531
SP3_SS3_CF5	(55,55)	(5,55)	(5,5)	(55,5)	(30,30)	30	0,9	3,53	878
SP3_SS4_CF1	(55,55)	(5,55)	(5,5)	(55,5)	(30,30)	40	0,5	3,18	209
SP3_SS4_CF2	(55,55)	(5,55)	(5,5)	(55,5)	(30,30)	40	0,6	3,34	240
SP3_SS4_CF3	(55,55)	(5,55)	(5,5)	(55,5)	(30,30)	40	0,7	3,22	302
SP3_SS4_CF4	(55,55)	(5,55)	(5,5)	(55,5)	(30,30)	40	0,8	3,33	420
SP3_SS4_CF5	(55,55)	(5,55)	(5,5)	(55,5)	(30,30)	40	0,9	3,22	662
SP3_SS5_CF1	(55,55)	(5,55)	(5,5)	(55,5)	(30,30)	50	0,5	3,06	211
SP3_SS5_CF2	(55,55)	(5,55)	(5,5)	(55,5)	(30,30)	50	0,6	3,56	319
SP3_SS5_CF3	(55,55)	(5,55)	(5,5)	(55,5)	(30,30)	50	0,7	3,54	330
SP3_SS5_CF4	(55,55)	(5,55)	(5,5)	(55,5)	(30,30)	50	0,8	3,39	628
SP3_SS5_CF5	(55,55)	(5,55)	(5,5)	(55,5)	(30,30)	50	0,9	3,27	826

Appendix B4 - SP4 Configuration Parameters

Figure 23: HOPSPACK configuration parameters for SP4 scenarios

Scenario	Initial Well Placement					Initial Step Size	Contraction Factor	Ultimate Recovery	Number of Iterations
[-]	[INJ1(X,Y)]	[PROD1(X,Y)]	[PROD2(X,Y)]	[PROD3(X,Y)]	[PROD4(X,Y)]	[grid cells]	[-]	MSm3	[-]
SP4_SS1_CF1	(55,5)	(55,55)	(5,55)	(5,5)	(30,30)	10	0,5	3,35	144
SP4_SS1_CF2	(55,5)	(55,55)	(5,55)	(5,5)	(30,30)	10	0,6	3,41	272
SP4_SS1_CF3	(55,5)	(55,55)	(5,55)	(5,5)	(30,30)	10	0,7	3,31	255
SP4_SS1_CF4	(55,5)	(55,55)	(5,55)	(5,5)	(30,30)	10	0,8	3,32	366
SP4_SS1_CF5	(55,5)	(55,55)	(5,55)	(5,5)	(30,30)	10	0,9	3,44	653
SP4_SS2_CF1	(55,5)	(55,55)	(5,55)	(5,5)	(30,30)	20	0,5	3,36	134
SP4_SS2_CF2	(55,5)	(55,55)	(5,55)	(5,5)	(30,30)	20	0,6	3,41	311
SP4_SS2_CF3	(55,5)	(55,55)	(5,55)	(5,5)	(30,30)	20	0,7	3,42	247
SP4_SS2_CF4	(55,5)	(55,55)	(5,55)	(5,5)	(30,30)	20	0,8	3,42	437
SP4_SS2_CF5	(55,5)	(55,55)	(5,55)	(5,5)	(30,30)	20	0,9	3,41	679
SP4_SS3_CF1	(55,5)	(55,55)	(5,55)	(5,5)	(30,30)	30	0,5	3,40	198
SP4_SS3_CF2	(55,5)	(55,55)	(5,55)	(5,5)	(30,30)	30	0,6	3,55	304
SP4_SS3_CF3	(55,5)	(55,55)	(5,55)	(5,5)	(30,30)	30	0,7	3,39	293
SP4_SS3_CF4	(55,5)	(55,55)	(5,55)	(5,5)	(30,30)	30	0,8	3,40	505
SP4_SS3_CF5	(55,5)	(55,55)	(5,55)	(5,5)	(30,30)	30	0,9	3,56	874
SP4_SS4_CF1	(55,5)	(55,55)	(5,55)	(5,5)	(30,30)	40	0,5	3,25	283
SP4_SS4_CF2	(55,5)	(55,55)	(5,55)	(5,5)	(30,30)	40	0,6	3,42	298
SP4_SS4_CF3	(55,5)	(55,55)	(5,55)	(5,5)	(30,30)	40	0,7	3,37	242
SP4_SS4_CF4	(55,5)	(55,55)	(5,55)	(5,5)	(30,30)	40	0,8	3,42	440
SP4_SS4_CF5	(55,5)	(55,55)	(5,55)	(5,5)	(30,30)	40	0,9	3,42	668
SP4_SS5_CF1	(55,5)	(55,55)	(5,55)	(5,5)	(30,30)	50	0,5	3,42	283
SP4_SS5_CF2	(55,5)	(55,55)	(5,55)	(5,5)	(30,30)	50	0,6	3,56	311
SP4_SS5_CF3	(55,5)	(55,55)	(5,55)	(5,5)	(30,30)	50	0,7	3,38	355
SP4_SS5_CF4	(55,5)	(55,55)	(5,55)	(5,5)	(30,30)	50	0,8	3,56	467
SP4_SS5_CF5	(55,5)	(55,55)	(5,55)	(5,5)	(30,30)	50	0,9	3,57	1044

Appendix B5 - SP5 Configuration Parameters

Figure 24: HOPSPACK configuration parameters for SP5 scenarios

Scenario	Initial Well Placement					Initial Step Size	Contraction Factor	Ultimate Recovery	Number of Iterations
[-]	[INJ1(X,Y)]	[PROD1(X,Y)]	[PROD2(X,Y)]	[PROD3(X,Y)]	[PROD4(X,Y)]	[grid cells]	[-]	MSm3	[-]
SP5_SS1_CF1	(30,30)	(15,15)	(45,15)	(45,45)	(15,45)	10	0,5	3,44	270
SP5_SS1_CF2	(30,30)	(15,15)	(45,15)	(45,45)	(15,45)	10	0,6	3,44	239
SP5_SS1_CF3	(30,30)	(15,15)	(45,15)	(45,45)	(15,45)	10	0,7	3,45	208
SP5_SS1_CF4	(30,30)	(15,15)	(45,15)	(45,45)	(15,45)	10	0,8	3,45	389
SP5_SS1_CF5	(30,30)	(15,15)	(45,15)	(45,45)	(15,45)	10	0,9	3,44	634
SP5_SS2_CF1	(30,30)	(15,15)	(45,15)	(45,45)	(15,45)	20	0,5	3,30	326
SP5_SS2_CF2	(30,30)	(15,15)	(45,15)	(45,45)	(15,45)	20	0,6	3,14	180
SP5_SS2_CF3	(30,30)	(15,15)	(45,15)	(45,45)	(15,45)	20	0,7	3,29	340
SP5_SS2_CF4	(30,30)	(15,15)	(45,15)	(45,45)	(15,45)	20	0,8	3,27	453
SP5_SS2_CF5	(30,30)	(15,15)	(45,15)	(45,45)	(15,45)	20	0,9	3,30	641
SP5_SS3_CF1	(30,30)	(15,15)	(45,15)	(45,45)	(15,45)	30	0,5	3,52	257
SP5_SS3_CF2	(30,30)	(15,15)	(45,15)	(45,45)	(15,45)	30	0,6	3,55	306
SP5_SS3_CF3	(30,30)	(15,15)	(45,15)	(45,45)	(15,45)	30	0,7	3,44	389
SP5_SS3_CF4	(30,30)	(15,15)	(45,15)	(45,45)	(15,45)	30	0,8	3,56	403
SP5_SS3_CF5	(30,30)	(15,15)	(45,15)	(45,45)	(15,45)	30	0,9	3,51	659
SP5_SS4_CF1	(30,30)	(15,15)	(45,15)	(45,45)	(15,45)	40	0,5	3,50	310
SP5_SS4_CF2	(30,30)	(15,15)	(45,15)	(45,45)	(15,45)	40	0,6	3,55	233
SP5_SS4_CF3	(30,30)	(15,15)	(45,15)	(45,45)	(15,45)	40	0,7	3,56	270
SP5_SS4_CF4	(30,30)	(15,15)	(45,15)	(45,45)	(15,45)	40	0,8	3,56	385
SP5_SS4_CF5	(30,30)	(15,15)	(45,15)	(45,45)	(15,45)	40	0,9	3,56	619
SP5_SS5_CF1	(30,30)	(15,15)	(45,15)	(45,45)	(15,45)	50	0,5	3,53	286
SP5_SS5_CF2	(30,30)	(15,15)	(45,15)	(45,45)	(15,45)	50	0,6	3,51	346
SP5_SS5_CF3	(30,30)	(15,15)	(45,15)	(45,45)	(15,45)	50	0,7	3,55	247
SP5_SS5_CF4	(30,30)	(15,15)	(45,15)	(45,45)	(15,45)	50	0,8	3,56	442
SP5_SS5_CF5	(30,30)	(15,15)	(45,15)	(45,45)	(15,45)	50	0,9	3,51	713

Appendix B6 - SP6 Configuration Parameters

Figure 25: HOPSPACK configuration parameters for SP6 scenarios

Scenario	Initial Well Placement					Initial Step Size	Contraction Factor	Ultimate Recovery	Number of Iterations
[-]	[INJ1(X,Y)]	[PROD1(X,Y)]	[PROD2(X,Y)]	[PROD3(X,Y)]	[PROD4(X,Y)]	[grid cells]	[-]	MSm3	[-]
SP6_SS1_CF1	(30,30)	(10,10)	(50,10)	(50,50)	(10,50)	10	0,5	3,51	186
SP6_SS1_CF2	(30,30)	(10,10)	(50,10)	(50,50)	(10,50)	10	0,6	3,51	122
SP6_SS1_CF3	(30,30)	(10,10)	(50,10)	(50,50)	(10,50)	10	0,7	3,51	206
SP6_SS1_CF4	(30,30)	(10,10)	(50,10)	(50,50)	(10,50)	10	0,8	3,51	249
SP6_SS1_CF5	(30,30)	(10,10)	(50,10)	(50,50)	(10,50)	10	0,9	3,51	412
SP6_SS2_CF1	(30,30)	(10,10)	(50,10)	(50,50)	(10,50)	20	0,5	3,51	273
SP6_SS2_CF2	(30,30)	(10,10)	(50,10)	(50,50)	(10,50)	20	0,6	3,51	187
SP6_SS2_CF3	(30,30)	(10,10)	(50,10)	(50,50)	(10,50)	20	0,7	3,57	235
SP6_SS2_CF4	(30,30)	(10,10)	(50,10)	(50,50)	(10,50)	20	0,8	3,51	259
SP6_SS2_CF5	(30,30)	(10,10)	(50,10)	(50,50)	(10,50)	20	0,9	3,51	413
SP6_SS3_CF1	(30,30)	(10,10)	(50,10)	(50,50)	(10,50)	30	0,5	3,53	147
SP6_SS3_CF2	(30,30)	(10,10)	(50,10)	(50,50)	(10,50)	30	0,6	3,51	132
SP6_SS3_CF3	(30,30)	(10,10)	(50,10)	(50,50)	(10,50)	30	0,7	3,51	161
SP6_SS3_CF4	(30,30)	(10,10)	(50,10)	(50,50)	(10,50)	30	0,8	3,51	232
SP6_SS3_CF5	(30,30)	(10,10)	(50,10)	(50,50)	(10,50)	30	0,9	3,51	445
SP6_SS4_CF1	(30,30)	(10,10)	(50,10)	(50,50)	(10,50)	40	0,5	3,51	226
SP6_SS4_CF2	(30,30)	(10,10)	(50,10)	(50,50)	(10,50)	40	0,6	3,51	168
SP6_SS4_CF3	(30,30)	(10,10)	(50,10)	(50,50)	(10,50)	40	0,7	3,51	215
SP6_SS4_CF4	(30,30)	(10,10)	(50,10)	(50,50)	(10,50)	40	0,8	3,51	237
SP6_SS4_CF5	(30,30)	(10,10)	(50,10)	(50,50)	(10,50)	40	0,9	3,51	440
SP6_SS5_CF1	(30,30)	(10,10)	(50,10)	(50,50)	(10,50)	50	0,5	3,51	188
SP6_SS5_CF2	(30,30)	(10,10)	(50,10)	(50,50)	(10,50)	50	0,6	3,51	115
SP6_SS5_CF3	(30,30)	(10,10)	(50,10)	(50,50)	(10,50)	50	0,7	3,50	174
SP6_SS5_CF4	(30,30)	(10,10)	(50,10)	(50,50)	(10,50)	50	0,8	3,51	279
SP6_SS5_CF5	(30,30)	(10,10)	(50,10)	(50,50)	(10,50)	50	0,9	3,51	441

Appendix B7 - SP7 Configuration Parameters

Figure 26: HOPSPACK configuration parameters for SP7 scenarios

Scenario	Initial Well Placement					Initial Step Size	Contraction Factor	Ultimate Recovery	Number of Iterations
[-]	[INJ1(X,Y)]	[PROD1(X,Y)]	[PROD2(X,Y)]	[PROD3(X,Y)]	[PROD4(X,Y)]	[grid cells]	[-]	MSm3	[-]
SP7_SS1_CF1	(30,30)	(5,5)	(55,5)	(55,55)	(5,55)	10	0,5	3,56	213
SP7_SS1_CF2	(30,30)	(5,5)	(55,5)	(55,55)	(5,55)	10	0,6	3,53	261
SP7_SS1_CF3	(30,30)	(5,5)	(55,5)	(55,55)	(5,55)	10	0,7	3,56	239
SP7_SS1_CF4	(30,30)	(5,5)	(55,5)	(55,55)	(5,55)	10	0,8	3,51	517
SP7_SS1_CF5	(30,30)	(5,5)	(55,5)	(55,55)	(5,55)	10	0,9	3,50	718
SP7_SS2_CF1	(30,30)	(5,5)	(55,5)	(55,55)	(5,55)	20	0,5	3,38	243
SP7_SS2_CF2	(30,30)	(5,5)	(55,5)	(55,55)	(5,55)	20	0,6	3,57	280
SP7_SS2_CF3	(30,30)	(5,5)	(55,5)	(55,55)	(5,55)	20	0,7	3,48	259
SP7_SS2_CF4	(30,30)	(5,5)	(55,5)	(55,55)	(5,55)	20	0,8	3,56	332
SP7_SS2_CF5	(30,30)	(5,5)	(55,5)	(55,55)	(5,55)	20	0,9	3,57	765
SP7_SS3_CF1	(30,30)	(5,5)	(55,5)	(55,55)	(5,55)	30	0,5	3,42	192
SP7_SS3_CF2	(30,30)	(5,5)	(55,5)	(55,55)	(5,55)	30	0,6	3,33	295
SP7_SS3_CF3	(30,30)	(5,5)	(55,5)	(55,55)	(5,55)	30	0,7	3,36	283
SP7_SS3_CF4	(30,30)	(5,5)	(55,5)	(55,55)	(5,55)	30	0,8	3,34	603
SP7_SS3_CF5	(30,30)	(5,5)	(55,5)	(55,55)	(5,55)	30	0,9	3,56	739
SP7_SS4_CF1	(30,30)	(5,5)	(55,5)	(55,55)	(5,55)	40	0,5	3,25	211
SP7_SS4_CF2	(30,30)	(5,5)	(55,5)	(55,55)	(5,55)	40	0,6	3,27	244
SP7_SS4_CF3	(30,30)	(5,5)	(55,5)	(55,55)	(5,55)	40	0,7	3,56	282
SP7_SS4_CF4	(30,30)	(5,5)	(55,5)	(55,55)	(5,55)	40	0,8	3,56	490
SP7_SS4_CF5	(30,30)	(5,5)	(55,5)	(55,55)	(5,55)	40	0,9	3,51	968
SP7_SS5_CF1	(30,30)	(5,5)	(55,5)	(55,55)	(5,55)	50	0,5	3,36	223
SP7_SS5_CF2	(30,30)	(5,5)	(55,5)	(55,55)	(5,55)	50	0,6	3,34	257
SP7_SS5_CF3	(30,30)	(5,5)	(55,5)	(55,55)	(5,55)	50	0,7	3,55	264
SP7_SS5_CF4	(30,30)	(5,5)	(55,5)	(55,55)	(5,55)	50	0,8	3,51	432
SP7_SS5_CF5	(30,30)	(5,5)	(55,5)	(55,55)	(5,55)	50	0,9	3,57	681

Appendix C - Optimiser-Simulator Interface

```
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <sstream>
#include <cmath>

using namespace std;

/* ------------------------------------------------------------------
 *  Platform/build specific symbols
 * ------------------------------------------------------------------
 */
#if defined(_WIN32)
  #if ((_MSC_VER == 1400) || (_MSC_VER == 1500))
    //----- WINDOWS MSVC COMPILER INSISTS THAT SECURE STRING FNS BE USED.
    #define HAVE_MSVC_SECURE_STRING_FNS
  #endif
#endif

/* ------------------------------------------------------------------
 *  Include file (WELLPOS.INC) editor
 * ------------------------------------------------------------------
 */
static bool writeNewIncludeFile(vector<double> wellPos)
{
    ofstream outfile ("WELLPOS.INC");

    // WRITING WELSPECS DATA
    outfile << "WELSPECS" << endl;
    outfile << "INJ1\tG1\t" << wellPos[0] << "\t" << wellPos[1]
        << "\t1715\tWAT /" << endl;
    outfile << "PROD1\tG2\t" << wellPos[2] << "\t" << wellPos[3]
        << "\t1715\tOIL /" << endl;
    outfile << "PROD2\tG2\t" << wellPos[4] << "\t" << wellPos[5]
        << "\t1715\tOIL /" << endl;
    outfile << "PROD3\tG2\t" << wellPos[6] << "\t" << wellPos[7]
        << "\t1715\tOIL /" << endl;
    outfile << "PROD4\tG2\t" << wellPos[8] << "\t" << wellPos[9]
        << "\t1715\tOIL /" << endl;
    outfile << "/" << endl;

    // WRITING WELSPECS DATA
    outfile << "COMPDAT" << endl;
    outfile << "INJ1\t" << wellPos[0] << "\t" << wellPos[1]
        << "\t1\t1\tOPEN\t1\t1*\t0.1905\t4*\t1* /" << endl;
    outfile << "PROD1\t" << wellPos[2] << "\t" << wellPos[3]
        << "\t1\t1\tOPEN\t1\t1*\t0.1905\t4*\t1* /" << endl;
    outfile << "PROD2\t" << wellPos[4] << "\t" << wellPos[5]
        << "\t1\t1\tOPEN\t1\t1*\t0.1905\t4*\t1* /" << endl;
    outfile << "PROD3\t" << wellPos[6] << "\t" << wellPos[7]
        << "\t1\t1\tOPEN\t1\t1*\t0.1905\t4*\t1* /" << endl;
    outfile << "PROD4\t" << wellPos[8] << "\t" << wellPos[9]
        << "\t1\t1\tOPEN\t1\t1*\t0.1905\t4*\t1* /" << endl;
    outfile << "/" << endl;

    outfile.close();

    return( 0 );
}
```

```
73  /* ------------------------------------------------------------------------
74   *  ECLIPSE simulation launcher
75   * ------------------------------------------------------------------------
76   */
77  static bool runEclipseSim()
78  {
79      remove("SIMPLECASE.RSM");
80
81      system("$eclipse SIMPLECASE");
82
83      return( 0 );
84  }
85
86  /* ------------------------------------------------------------------------
87   *  RSM file reader that extracts last FOPT value
88   * ------------------------------------------------------------------------
89   */
90  double readRSMFile(const string rsmFileName)
91  {
92      double foptLast = -1;
93
94      ifstream file(rsmFileName + ".RSM");
95
96      if (file)
97      {
98          string lastLine;
99          while(file >> ws && getline(file, lastLine)){}
100         string buf;
101         stringstream ss(lastLine);
102         vector<string> tokens;
103         while (ss >> buf)
104         {
105             tokens.push_back(buf);
106         }
107         if (!tokens.empty())
108         {
109             foptLast = stod(tokens.end()[-2]); // Get second to last element
110         }
111     }
112     else
113     {
114         cout << "Unable to open .RSM file." << endl;
115     }
116
117     return( foptLast );
118 }
119
120 /* ------------------------------------------------------------------------
121  *  Keep track of a single run and record it to a file
122  * ------------------------------------------------------------------------
123  */
124 static bool writeSummaryToFile(vector<double> wellPos, double foptLast)
125 {
126
127     ofstream sumfile ("RUNSUM.TXT", std::ios::app);
128
129     sumfile << "========== NEW ITERATION ==========" << endl;
130     sumfile << "=== WELL COORDINATES ===" << endl;
131     sumfile << "Well\tX\tY" << endl;
132     sumfile << "INJ1\t" << wellPos[0] << "\t" << wellPos[1] << endl;
133     sumfile << "PROD1\t" << wellPos[2] << "\t" << wellPos[3] << endl;
134     sumfile << "PROD2\t" << wellPos[4] << "\t" << wellPos[5] << endl;
135     sumfile << "PROD3\t" << wellPos[6] << "\t" << wellPos[7] << endl;
136     sumfile << "PROD4\t" << wellPos[8] << "\t" << wellPos[9] << endl;
137     sumfile << "=== FOPT ===" << endl;
138     sumfile << "FOPT = " << foptLast << endl;
139     sumfile << "========== END ITERATION ==========" << endl;
140     sumfile << "" << endl;
141
142     sumfile.close();
143
144     return( 0 );
145 }
146
```

```
147 /* ------------------------------------------------------------
148  *  Internal Function read_input_file
149  * ------------------------------------------------------------
150  */
151
152 static bool  readInputFile (const string &szExeName,
153                             const string &szFileName,
154                             vector<double> &wellPos)
155 {
156     const int varNum = 10; // Declare the number of variables
157     const int REQBUFLEN = 10;
158     char reqBuf[REQBUFLEN + 1];
159
160     ifstream fp;
161
162     //----- OPEN THE INPUT FILE.
163     fp.open (szFileName.c_str());
164     if (!fp)
165     {
166         cerr << szExeName << " - ERROR opening input file '"
167              << szFileName << "'." << endl;
168         return( -1 );
169     }
170
171     //----- READ AND VERIFY THE INPUT REQUEST TYPE.
172     fp >> reqBuf;
173     for (int  i = 0; i < REQBUFLEN + 1; i++)
174     {
175         if ((reqBuf[i] == '\n') || (reqBuf[i] == '\r'))
176         {
177             reqBuf[i] = 0;
178             break;
179         }
180     }
181     if ((reqBuf[0] != 'F') || (reqBuf[1] != 0))
182     {
183         cerr << szExeName << " - ERROR reading request type from '"
184              << szFileName << "'." << endl;
185         cerr << "  Read '" << reqBuf << "'" << endl;
186         fp.close();
187         return( -1 );
188     }
189
190     //----- READ THE LENGTH OF x.
191     int  n;
192     fp >> n;
193     if (n != varNum)
194     {
195         cerr << szExeName << " - ERROR reading n from '"
196              << szFileName << "'." << endl;
197         cerr << "  Read " << n << ", but problem size should be " << varNum << endl;
198         fp.close();
199         return( -1 );
200     }
201
202     wellPos.resize (n);
203
204     //----- READ x.
205     for (int  i = 0; i < n; i++)
206     {
207         fp >> wellPos[i];
208         cout << round(wellPos[i]) << "\t";
209     }
210     cout << "" << endl;
211     fp.close();
212
213     //----- RETURN SUCCESS.
214     return( 0 );
215 }
216
217
218
219
220
221
```

```
/* ------------------------------------------------------------------
 *  Internal Function write_output_file
 * ------------------------------------------------------------------
 */
static bool  writeOutputFile (const string & szExeName,
                              const string & szFileName,
                              const double f)
{
    ofstream  fp;

    fp.open (szFileName.c_str(), ios::trunc);
    if (!fp)
    {
        cerr << szExeName << " - ERROR opening output file '"
             << szFileName << "'." << endl;
        return( -1 );
    }

    //---- WRITE THE NUMBER OF OBJECTIVES AND THEIR VALUES TO THE OUTPUT.
    fp << "1" << endl;
    fp.precision (15);      //-- WRITE ALL DIGITS
    fp << f << endl;

    fp.close();

    return( 0 );
}
```

```
/* ----------------------------------------------------------
 *  Main routine for evaluation executable.
 *
 *  Each execution reads a vector from a file, evaluates the objective
 *  function, and writes the function value to an output file.  HOPSPACK
 *  generates the file names and decides when to invoke this executable.
 *  If there is an error, either create no output file, or create an
 *  output file with an error message.
 *
 *  @param argc  Number of command line arguments.
 *  @param argv  Command line arguments (input and output file names).
 *  @return      0 if successful.
 * ----------------------------------------------------------
 */
int  main (const int argc, const char * const argv[])
{
    vector<double> wellPos;
    double foptLast = -1;
    int nRetStatus;

    /* CHECK THE COMMAND LINE ARGUMENTS.
     * IN THIS EXAMPLE ONLY THE INPUT AND OUTPUT FILE NAMES ARE USED.
     */
    if (argc != 5)
    {
        fprintf (stderr, "usage: %s <input file> <output file> <tag> <type>\n",
                 argv[0]);
        return( -100 );
    }

    // Read an internal input file
    nRetStatus = readInputFile (argv[0], argv[1], wellPos);
    if (nRetStatus != 0)
        return( nRetStatus );

    for (unsigned int  i = 0; i < wellPos.size(); i++)
    {
        wellPos[i] = round(wellPos[i]);
    }

    // Using wellPos values to write a new WELLPOS.INC file
    nRetStatus = writeNewIncludeFile(wellPos);
    if (nRetStatus != 0)
        return( nRetStatus );

    do
    {
        // Run ECLIPSE simulations
        nRetStatus = runEclipseSim();
        if (nRetStatus != 0)
            return( nRetStatus );

        // Reading RSM file to extract last FOPT value
        foptLast = readRSMFile("SIMPLECASE");
        if (foptLast != -1)
        {
            // Write summary of a run to a file
            nRetStatus = writeSummaryToFile(wellPos, foptLast);
            if (nRetStatus != 0)
                return( nRetStatus );
        }
    } while( foptLast == -1 );

    // Write an internal output file
    nRetStatus = writeOutputFile (argv[0], argv[2], foptLast);

    return( nRetStatus );
}
```