OPTIMISATION OF COMPLEX SIMULATION MODELS

by

Cécile Bezuidenhoudt

Thesis presented for the degree of Master of Science in

Department of Statistical Science

University of Cape Town

February 2013

Supervised by Dr Ian Durbach and Prof Theodor Stewart

The financial assistance of the National Research Foundation (NRF) and the Institute of Applied Statistics towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to the NRF or the Institute of Applied Statistics.

Plagiarism Declaration

I know the meaning of plagiarism and declare that all of the work in the thesis, save for that which is properly acknowledged, is my own.

Signature:

Date:

Abstract

Computer simulation models are widely and frequently used to model real systems to predict output responses under specified input conditions. Choosing optimal simulation parameters leads to improved operation of the model but it is still a challenge as to how to go about optimally selecting these parameter values. The aim of this thesis was to see if a method could be found to optimise a simulation model provided by a client. This thesis provides a review of the literature of various simulation optimisation techniques that exist. Five of these simulation optimisation techniques - Simulated Annealing, Genetic Algorithms, Nested Partitions, Ordinal Optimisation and the Nelson-Matejcik Method - were selected and applied to a test case stochastic simulation model to gain an understanding into the techniques for their use in optimising the test model. These techniques were then used and applied to optimise a real life simulation model provided by a client. A technique combining the Ordinal Optimisation and Simulated Annealing optimisation methods provided the best results. This technique was provided to the client as a strategy to implement into their simulation model.

Acknowledgments

Firstly, I would like to express my gratitude to my supervisors: Dr Ian Durbach and Prof Theodor Stewart. Thank you for your guidance, time and patience. It was a great privalege to work with you.

Thank you to my friends and family for the love, support and encouragement that they have provided for the last two years, especially in the last few months of putting this thesis together.

Thank you to the client for providing the real life simulation model that I used. Particular thanks goes to the members of their team who not only helped me create the link between the programme that ran the simulation model and the programme that ran the optimisation method but who were also so willing to answer all my questions.

Thank you to the Institute of Applied Statistics who provided not only funding throughout this two year process but also gave me a wonderful opportunity to speak to high school children about the great future of the field of statistics and operations research and how, if they wanted to, they could be a part of it.

Thank you to the National Research Fund who provided funds for the second year of my master's degree.

Contents

List of Figures

List of Tables

Chapter 1. Introduction

1.1 Introduction

Computer simulation models are widely and frequently used as models of real systems to evaluate output responses. By choosing the optimal simulation parameters, it can lead to improved operation but it is still a challenge as to how to go about selecting these parameter values. The goal of this thesis is to apply selected simulation optimisation techniques to a test case simulation model in order to gain an understanding of the way these techniques operate. This knowledge is then used when trying to optimise a real world simulation model provided by a client with the aim of establishing which of the optimisation techniques tested is best suited to provide the optimum result.

1.1.1 Overview of Problem Setting

We live in an environment where there are many systems around us. These systems, which vary in level of complexity, can be defined as a collection of entities that act and interact dynamically toward the accomplishment of some logical end. An example of a system would be how much milk to keep in a store. The entities that act and interact together would be the individuals who buy the milk, the delivery of the milk, the different types of milk, the shelf life of the milk etc. All these interactions would have an impact on the amount of milk in a store at any one time.

There are many lenses through which a system can be studied, from looking at the real life system to simplified replications of the system. These lenses are outlined in Figure 1.1 [66]. A system can be studied by experimenting with the actual system or by experimenting with a model of the actual system, which itself is broken down into further categories. Continuing with the milk example, this dynamic system could be studied by going into a store and monitoring the system, creating a 'made-up' store and monitoring how the system functions or creating a computer simulation model which models the system. Each of these methods of studying the system has its advantages and disadvantages and certain methods might be more suitable than others given the situation.

Although there are many ways in which systems can be studied, studying them through simulation models is often very desirable. This is due to the many advantages that simulation models have such as their ability to compress or expand time, their ability to control sources of variation, they are able to avoid errors in measurement, they can be stopped, reviewed and can

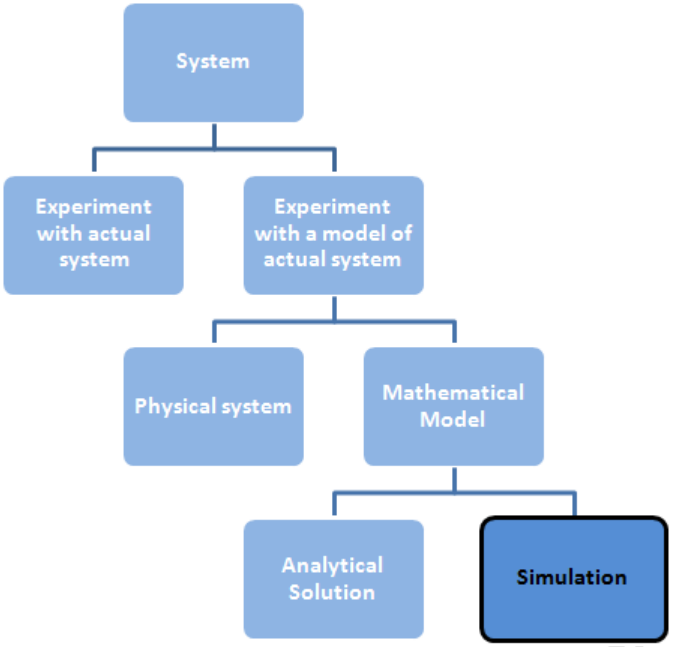

Figure 1.1: System analysis breakdown [66]

facilitate replications and, lastly, the modeler can control some level of detail. With the milk example, all these features serve in favour as to why simulation would be an ideal way to model the system. For example, the simulation model is good at compressing time and therefore one month of the actual system could be modeled in a few minutes using powerful computers. Also, many of the systems that model the amount of milk in the store could be run at the same time which would allow more information to be recored. Modeling the system using simulation also acts as a cost saving as observing a computer simulation is cheaper than observing the real system.

The goal of simulation models is to check the design of the system and provide a model that can be studied and hence, the properties concerning the behaviour of the actual system or its subsystem can be inferred. In the broadest sense, simulation is a tool to evaluate the performance of a system, existing or proposed, under different configurations of interest and over long periods of real time [69].

One of the major features of using simulation models is that one can change the inputs of a simulation model easily and try to observe the system performance under different sets of input values. In this way a 'what-if' analysis can be performed; this means that the input values can be set and the simulation run. From this, the output can be analysed and used to help answer the question of 'what would happen in the system if the inputs were defined in such a way?' In the example, this could be done by changing the amount of milk that is ordered and monitoring the impact this had on the system, similarly, one could only stock one type of milk or decrease the amount of milk bought. All of which are factors that would affect the amount of milk that was in the store.

Therefore, it is natural to try to find the set of inputs, within their constrained spaces, that optimises the system performance. In this case, the optimum is achieved when the optimum value of a predefined goal is found. This is where the concept of simulation optimisation comes in. By choosing the optimal simulation parameters, it can lead to improved operation but it still remains a challenge as to how to select these parameter values. What this means for the milk example is that through applying simulation optimisation to the model, one would be able to find the input values that enable the optimum amount of milk to be kept at the store.

The simulation optimisation problem setting has been defined in [31] to have three primary components: input and output variables, an objective function and constraints. The objective function and constraints can involve both the input and output variables, and either (or both) can involve stochastic components. Since the output variables are simulation model performance measures, they are quantitative in nature, such as the quantity of milk in a store at a given time. However, unlike standard mathematical programs, the input 'variables' may be either quantitative or qualitative. An example of a quantitative input would be the amount of milk delivered while a qualitative one would be the season of the year. In general, there is a single objective function. Multiple performance measures are usually handled by combining them into the objective function using appropriate weights, or by including them as constraints. Constraints can be further subdivided into explicit versus implicit, and deterministic versus stochastic. There are many constraints in this example such as the storage capacity of the store and the amount of milk that can be delivered at one time.

Simulation optimisation is difficult because not only is there an optimisation setting but there is now the added complexity that the output values are stochastic in nature due to the fact that they are generated from the simulation model. A summary of this difficulty is given in [11]: 'Even when there is no uncertainty, optimisation can be very difficult if the number of design variables is large, the problem contains a diverse collection of design variable types, and little is known about the structure of the performance function. Optimisation via simulation adds an additional complication because the performance of a particular design cannot be evaluated exactly, but instead must be estimated. Because we have estimates, it may not be possible to conclusively determine if one design is better than another, frustrating optimisation algorithms that try to move in improving directions. In principle, one can eliminate this complication by making so many replications, or such long runs, at each design point that the performance estimate has essentially no variance. In practice, this could mean that very few alternative designs will be explored due to the time required to simulate each one.'

At this point it is worth clarifying the difference between a deterministic simulation setting and a stochastic simulation setting. In a deterministic simulation setting, there is a known set of inputs which will results in a unique set of outputs. In a sense, this means that the assumptions and equations you select determine the results. The only way in which the output can change is due to a change in an assumption or equation. In a stochastic simulation setting, however, there are one or more random variables as inputs. The fact that there are random inputs means that there will then be random outputs. Since the outputs are random, they can be considered only as estimates of the true characteristics of a model. In a stochastic simulation, the output

measures must be treated as statistical estimates of the true characteristic of the system.

Dealing with a stochastic simulation model as opposed to a deterministic simulation model in general, and more specifically in an optimisation setting, therefore adds a level of complexity to the simulation optimisation framework. This is because not only is there additional complications due to the simulation of a real life system as explained in [11] above, but there is now also the added complication of the random variables used as inputs and therefore there is more uncertainty present making the generation of a solution a more complex task.

The challenge faced in the milk example can be defined as follows. One would need to find the best method which generates the best input values. These input values would then lead to the most optimum amount of milk kept at the store.

Many simulation optimisation techniques have been established for this purpose. Some are more appropriate than others, yielding 'intelligent' trial-and-error methods of choice in how to run the simulation to find the best solution. The next thing to do is then to find the simulation optimisation technique that is best suited to the model at hand. Once equipped with this, it can be applied to the simulation model of interest and the results produced can be analysed.

1.1.2 Aims and Strategy

Now that the setting has been defined, the aim and strategy of this thesis can be defined in three main points as follows:

1. To demonstrate how a complex simulation model of resource planning can be optimised.
2. To compare, in the course of this demonstration, the relative performance of different simulation optimisation methods.
3. To assess the strengths and weaknesses of these methods; both generally in the simulation optimisation setting as well as specifically with reference to the complex simulation model being analysed.

1.1.3 Background to the Problem

In this case, a client approached us with a simulation model, built by the client, that was designed to model the way in which of a group of their plants function. This model takes in input values, such as stock delivery and quality of the stock, and from this produces outputs, such as the amount of stock that was used to be turned into the desired supply amount as well as the cost that it takes to produce the total supply. Through the running of the simulation model, an understanding of how their plants run was attained without having to look at the real life system and therefore saving on both time and money.

Although the client had a simulation model that captured the ins and outs of the performance of their plants they were somewhat restricted. This was due to the fact that they were not able to tell under what circumstances the plants would run optimally and, over and above that, what this optimum value would be. As such, they had a question that they wanted answered: how

to best run their plants so that their objectives are met in the best possible way? This meant that the client approached us with the desire to find a simulation optimisation technique that they could apply to their model so that they would be able to select the best input values to generate the best output values.

1.1.4 Objectives

This thesis seeks to address the practical implications of imposing optimisation techniques onto simulation models to try and find the best solution. The objective of this thesis is to provide a general overview of the primary simulation optimisation techniques in the research literature which include heuristic solution methods, gradient based search methods, statistical methods and response surface methods. From the simulation optimisation techniques discussed, focus will be placed on a selection of methods that are adapted from the derivative-free simulation optimisation techniques. These techniques are tested on a test case simulation model to gain further understanding into the way these methods operate. Lastly, a choice of simulation techniques is then applied to the real life simulation model provided by the client to try and come up with the optimum design which facilitates in meeting their goal while at the same time providing the strengths and weakness associated with the various simulation optimisation techniques tested.

1.1.5 Limitations

There are limitations associated with this thesis and these are listed below.

- The first limitation that is mentioned is that the field of simulation optimisation is a growing one and as a result new simulation optimisation techniques are constantly being developed. This thesis only focuses on a few of the simulation optimisation techniques but it is noted that there may be other better suited simulation optimisation techniques that are available than the ones that are explored in this thesis. The methods that were chosen were done so because they appeared well suited to the problem and, given the time constraints, the number of methods chosen was limitted.

- Secondly, a simulation optimisation technique is only as good as the model that you provide. This means that although the simulation optimisation techniques might produce good results and shows to be well suited to the problem at hand, if the simulation model itself has not been well designed then the results achieved have no significance. The real life simulation model that is being optimised in this thesis is designed by the client and as such it is assumed that it has been correctly designed to model the real life system.

- There is also a limitation present in terms of the computational power available; should better computers be used, the results that could be generated could be done so in a shorter amount of time or more accurate results could be generated in the same amount of time.

- It is also noted that the real life simulation model that is to be optimised was created by the client and therefore the way in which it functions, both in terms of the software used to create it as well as to the ins and outs of the model itself, was limited to what was provided by the client.

1.1.6 Plan of Development

This thesis is broken down into the following chapters:

- Chapter 2 introduces the concept of simulation optimisation in more detail and provides a review of the literature on this topic. This is done by firstly defining simulation optimisation. From this, the various simulation optimisation techniques that are to be discussed are classified based on their functioning and capabilities. There are four main classifications; heuristic methods, gradient-based search methods, statistical methods and lastly response surface methodology. A review of the literature of these methods is then provided.

- Chapter 3 introduces a test case stochastic simulation model that is created to test various simulation optimisation techniques. From the literature review, five simulation optimisation techniques that appear well suited to optimise the test case simulation model are selected to be applied as they appear to be the best suited to the achieve the optimisation of the test case simulation model. These five simulation optimisation techniques are Genetic Algorithm, Simulated Annealing, Nested Partitions, Nelson-Matejcik's procedure and lastly Ordinal Optimisation. These techniques are applied to the test case simulation model. This allows for a better understanding of the simulation model as well as provides indications towards the strengths and weaknesses of each method.

- In chapter 4, an overview of the simulation model provided by the client is given as well as a better definition of what the simulation optimisation of this model aims to achieve. The five simulation optimisation techniques that were applied to the test case simulation model are then applied to the real life simulation model where possible. From this, the weaknesses and strengths of these methods are noted and a method which is best suited to the optimisation of the simulation model is established.

- Lastly, chapter 5 provides an overview of the conclusions that were found and identifies areas of further research.

Chapter 2. Literature Review

2.1 Introduction

Two important concepts in Operations Research are simulation and optimisation. In recent years, the view of combining these two techniques, to form simulation optimisation, has received a lot of attention. The aim of this literature review is to focus on what in fact simulation optimisation is and then look at the various techniques available to perform this task.

Simulation, in itself, is a tool which is used to approximate reality with some purpose in mind. It allows fast, inexpensive and non-disruptive examination and testing of a large number of scenarios prior to actually implementing a particular decision in the real environment [14]. Simulation models can therefore avoid the main problems that real life scenarios or situations face such as being expensive and time consuming and the fact that they can often be of a questionable ethical nature and, at times, the system may not yet exist in a real-life context. Simulation models allow the user to construct a situation, regulate variables at both their starting values and by the way that they will be adjusted throughout the simulation, and then observe what happens to the system over a period of time. A few examples of areas in which simulation models have been used are technology, safety engineering and education.

A few advantages of simulation are that it is cost and time effective (to the extent that it is cheaper than running the system in real life although there are costs involved in terms of CPU usage and time used in running the simulation model) and allows us to observe potential outcomes under which there is minimal or no control. Simulation also helps in that the system can be replicated many times which is often not possible or highly expensive to perform in a real world context. This allows the range of potential outcomes to be observed, shows the way in which variables within the system interact with one another and which variables are the most important to the system. Lastly, simulation also helps describe the system which is useful in hypothesis testing and in predicting future behaviour.

There are also some disadvantages with simulation. A few are that building the simulation model is both an art and a science relying on both the quality of the model and the ability of the modeller. There is also the fact that the results obtained from simulation can, at times, be hard to interpret and can be easily misused in such that the information provided is not used correctly. Lastly, it can be expensive and time consuming to set up the simulation model. It is important to note that although simulation models can show how a system performs, it cannot provide in itself (without additional work) the best conditions under which the model

can perform.

Optimisation, in itself, is a completely separate topic to simulation. Simulation tells us how a system operates while optimisation looks for the 'best' answer or 'best' conditions under which a system performs. The reason for which 'best' is in inverted commas is due to the fact that the solution, in an Operations Research context, is the best solution available given the information and structure contained within and of a problem [14]. Optimisation is present in many areas and covers a wide variety of problems and can be used to optimise a single function or a system of functions. Examples are the optimisation of land use, the optimisation of an allocated budget and the optimisation of a mathematical function.

An advantage would be that optimisation allows the 'best' solution under which a system can be implemented to be found. A disadvantage is that the systems for which optima are required can be rather complex making optimisation rather difficult. Also, in order to apply an optimisation method the system which is to be optimised must be fully defined.

Now that the concepts of simulation and optimisation have been briefly introduced one can focus on joining these concepts together to form simulation optimisation. In simulation optimisation, two processes are combined where the one process' advantages compensate for the other process' disadvantages. The basic idea of simulation optimisation is to run a simulation which is an imitation of a real life system and then, from this, find the optimum values under which this system performs. Essentially, a feedback process is created whereby inputs are inserted into a simulation model from which the output is subjected to an optimisation strategy which then provides new inputs for the simulation model [18]. This process is demonstrated in Figure 2.1. In other words, a simulation optimisation problem is an optimisation problem where the objective function, constraints or both are responses that can only be evaluated by computer simulations. Ideally, one wants to minimise the resources spent while maximising the information obtained in a simulation experiment [9].

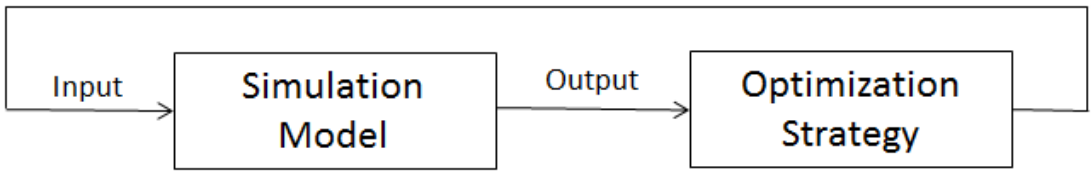

Figure 2.1: Feedback process of Simulation Optimisation Model

Simulation optimisation can be formally defined as to find a composition or design that minimises the objective function defined by

$$\min_{\theta \in \Theta} J(\theta) = E[L(\theta, \omega)]$$

where $\theta \in \Theta$ represents the (vector of) input variables, $J(\theta)$ is the objective function, ω represents the sample path (simulation replication) and L is the sample performance measure. Due to the fact that the performance of $J(\theta)$ cannot be evaluated analytically the expectation

of the performance L is taken. The constraint set Θ may be either explicitly given or implicitly defined [27]. Without loss of generality we have that we can turn this minimisation problem into a maximisation problem by multiplying the objective function by a negative 1. The minimisation setting will be used for the remainder of the literature review without loss of generality.

There are two things that need to be noted before moving on. Firstly, although the above definition is the one chosen for the remainder of the literature review, there are also other ways in which a system may be viewed as optimal, for example, minimising the dispersion of the response [9]. Secondly, when defining the simulation optimisation model, one assumes that the model has gone through a screening phase in which the factors that are considered unimportant are eliminated from the optimisation problem and all possible transformations of the factors and the response have already taken place [41].

In basic terms, the simulation optimisation concept is easy to understand but when looking a little more closely one realises that there are a few complications to which many opinions have been formed in the literature and techniques created to overcome these issues. A few of these complications are described as follows:

- To examine all possible solutions and choose the best one is very difficult and time consuming due to the large range of the parameter values and combinations of these parameters [14].
- There are many uncertainties and complexities modeled by the simulation and as a result often lead to the analyst not having a very clear idea of the shape of the response surface.
- There is also the issue that there can be many local maxima and therefore it can be hard to find the best solution if focusing on a specific neighbourhood (this is where normal optimisation techniques such as rank and selection, which will be discussed further on, fall short) [14].
- There exists the possibility that the search space is not compact and as such there could be zones of parameter values that are not allowed [81].
- Lastly, simulation optimisation methods are not always complete (in that it is not guaranteed for a finite set and finite time to find the optimal solution) but are generally approximate algorithms (in that there is a sacrifice of finding the optimal solution for being able to find a satisfactory solution in a given amount of time) [15].

The major advantages of simulation optimisation techniques, pointed out in [10], are:

- Even though the simulation models are often quite complex, the performance of the optimisation process is not significantly affected by this complexity.
- For stochastic systems, the variance of the response is controlled by various output analysis techniques.
- Where structural optimisation of systems are considered, simulation provides an advantage that is not often possible in classical optimisation procedures. This is that from one

iteration to the next the objective function and constraints can be changed based on the information that is obtained after each simulation run.

Other authors have stated additional advantages such as:

- The ability to handle a much larger number of scenarios than traditional optimisation approaches as well as many sources and types of uncertainty [14].
- The solution space can be intelligently searched without the need for the analyst's intervention [88].

[9] further defines four key aspects that one needs to take into account when dealing with simulation optimisation as opposed to normal optimisation. These are listed as follows:

- Differentiation or exact calculations of local gradients is not possible due to the fact that there does not exist an analytical expression of the objective function nor of the constraints.
- The objective function and constraints are stochastic functions of the deterministic decision variables. This in itself presents a major problem in estimation of even approximate local derivatives and acts as a disadvantage for using complete enumeration because based on just one observation at each point the best decision cannot be determined.
- Computer simulations are much more expensive to run than evaluating analytical functions which makes the efficiency of the optimisation algorithms more important.
- Most practitioners use some kind of language for modelling their systems. Optimisation requires using some other kind of programming language which differs from one practitioner to the next. Linking simulation models with generic optimisation routines is not always easy to achieve. Fu *et al.* [28] elaborate further on this saying that much research on simulation optimisation to date is concerned with methods that require a certain amount of sophistication on the part of the user in terms of understanding both the details of the optimisation approach being used and the nature of the stochastic processes underlying the simulation. In addition, in order to guarantee convergence these approaches are sometimes rather conservative which can lead to slow convergence in practice.

The disadvantages of the applying optimisation techniques to simulation models are listed below [77, 96]:

- When dealing with stochastic optimisation, in general, the best solution or partition found by the algorithm might not be the optimum solution or always contain the most optimum solution due to the inherent random nature of the problem at hand.
- When dealing with noisy objective functions, from a practical point of view, there is now additional noise introduced into the decision of determining which region should become the most promising region in the next iteration. Everything else being equal, this can

be expected to reduce the probability that the correct region is selected. The question of determining how much computational effort is needed to make the correct selection is therefore more important than before. Furthermore, the noise is often controllable to some extent by using more computational effort.

There are two main approaches for conducting simulation based optimisation [32]. The first is to carry out the simulation first and store these appropriately. This then converts a stochastic problem into a deterministic one based on a large enough set of samples to well approximate the desired problem. The second approach is to carry out a relatively small set of simulations and iteratively improve upon the current solution until a sufficiently good solution is reached. The first approach has become more popular with the fact that computer power has improved and the ability to then be able to use the tools that have been developed for deterministic optimisation. Most simulation models that are solved these days are done using a process whereby the inputs are controlled to get an output and there is no control on the actual output i.e. a forward method. Controlling the output to find correct inputs is a more complicated version to solve [81].

There is no single or standard approach to optimising a system where the data for the analysis are based on experiments conducted with a simulation model [42]. There are many available methods to use in simulation optimisation. Figure 2.2 shows the breakdown of the various simulation optimisation techniques that are to be discussed in this literature review. It is noted that the techniques mentioned are neither mutually exclusive or exhaustive of the simulation optimisation techniques but provide an outline of the most well known and used techniques.

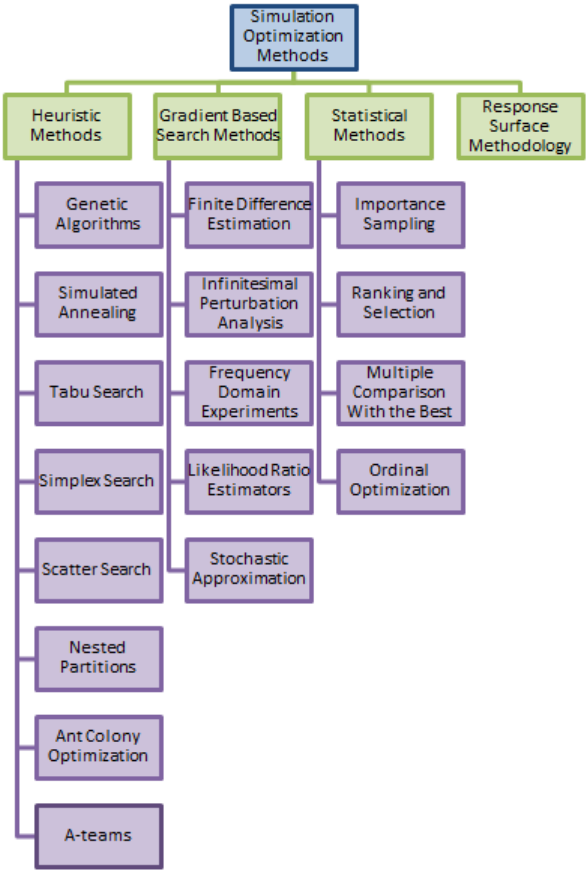

Figure 2.2: Breakdown of simulation optimisation techniques

The remainder of this literature review presents an overview of these various simulation optimisation techniques. For each method, an overview of the actual method is given, references of where a more detailed explanation of the method can be found and ends with at least one reference of a practical example of where the method has been implemented.

2.2 Heuristic Solution Methods

A heuristic is defined as a method which, on the basis of experience or judgments, seems likely to yield a reasonable solution to a problem but which cannot be guaranteed to produce the mathematically optimal solution [98].

Heuristic optimisation methods call for a reduced need for what can be seen as a rather restrictive set of assumptions required for other processes such as mathematical optimisation. Advantages for using heuristics defined in [98] are:

- They are relatively easy to implement.
- They provide fast results.

- They are robust to the extent that they can be less sensitive to variations in problem characteristics and data quality.
- Within the optimisation setting, they can provide good initial solutions to an iterative system, they can provide bounds which can be used to reduce the search space in partial enumeration optimisation methods and they can help guide a search process.

Instances where the use of heuristic methods is beneficial are [98]:

- Areas where a combinatorial explosion of the possible combinations of the decision variables that could be optimal arises.
- Situations where there is difficulty in evaluating the objective function (or in having probabilistic constraints) due to the presence of stochastic variables.
- Conditions that change markedly with time, which requires a whole time series of solutions rather than a single solution at a point in time.

There are a wide variety of heuristics that are available. The choice of heuristic depends on a number of factors. A few of these factors are [98]:

- Whether the decision area is strategic, tactical or operational.
- The frequency with which a decision is made.
- The development time available.
- The analytical qualifications of the decision makers involved.
- The size of the problem.
- The absence or presence of significant stochastic elements.

[98] describes seven basic 'types' of heuristic methods. These types, which will be described below, are not necessarily mutually exclusive.

1. *Randomly generated solutions* is a method which generates randomly feasible solutions to the problem and chooses the best one. This method is time consuming and it is not guaranteed that the optimum reached is globally optimal. It does, however, allow understanding of the model and the relationships that lie within it.
2. *Problem decomposition/partitioning* which allows you to take a complex problem and decomposes or partitions it into a number of sub-problems. This can be done through solving the subproblems independently, joining them together and finding one solution or solving the subproblems either sequentially or iteratively. This partitioning can be by natural hierarchy of decisions, by major resource or by chronological time of decision.

3. *Inductive methods* have two aspects to them. The first is generalisation from smaller versions of the same problem or a closely related problem. The second is where one analyses the case where one or more parameters take on very large values which allows insight for the more difficult case of intermediate values of parameters.

4. *Methods that reduce the solution space* essentially cut back on the number of solutions that are considered while hopefully not seriously affecting the quality of the solution obtained. This method is helpful as it can eliminate solutions that do not satisfy the conditions of the model by tightening existing constraints or including new constraints or considering solutions which satisfy a particular property.

5. *Approximation models* which are specifically concerned with manipulating the mathematical model in some way. There are four main ways in which this can be done: aggregation of parameters, modification/approximation of objective function, approximating probability distributions or stochastic processes and, lastly, changing the nature of the constraints including relaxation methods.

6. *Constructive methods* use the data of the problem to construct a solution. This is a step by step process that follows a set of rules defined prior to implementation. These rules concern the choice of the initial subcycle or starting points, a criterion to chooses the next element to add to the solution and the selection of the position where the new element will be inserted. Normally, the process has to be completed in order to get a solution to the problem.

7. *Local improvement (neighbourhood search) methods* are based on the concept of feasible solutions in the neighbourhood of the current solutions that are evaluated. If one of those solutions is better than the current solution it becomes the new current solution and its neighbourhood is examined etc. until no improvement can be found and the current solution then becomes the local optimum. The fundamental issue/weakness is that only a local optimum is guaranteed. Problems lie within defining exactly what the neighbourhood is and in deciding if you want to move to the first point that brings an improvement or if you want to perform an exhaustive search.

Now that the term heuristic is understood the concept of meta-heuristics follows. Meta-heuristics are essentially combinations of a different number of the heuristic methods mentioned above. The reason for the 'meta-' prefix is that metaheuristics do not specify all the details of the search which can thus be used by a local heuristic for a specific application. Instead, they specify general strategies to direct specific aspects of the search [77]. [78] defines a metaheuristic as "an iterative master process that guides and modifies the operations of subordinate heuristics to produce efficient high-quality solutions. It may combine intelligently different concepts to explore the search space using adaptive learning strategies and structured information."

Metaheuristics have provided a general solution strategy to many problems. There does, however, arise the issue of having to adjust the metaheuristic to then fit the actual problem

or problem class but this does require less work than actually having to develop a specific heuristic for a specific application [77]. Metaheuristics are particularly concerned with not getting stuck as a local optimum and/or reducing the search space. The metaheuristic based simulation optimisation framework is also very flexible in terms of the performance measures that the decision maker wishes to evaluate. The biggest hindrance is not performing the actual optimisation but rather carrying out the simulation model with the specified values of the decision variables (mainly due to the fact that this can take up a lot of computational time) [14].

The simple metaheuristic framework is defined as follows:

- Obtain an initial solution (solution set) θ_0 and set $k = 0$
- Repeat until the specified stopping criteria is satisfied:
  - Identify the neighbourhood $N(\theta_k)$ of the current solution(s)
  - Select candidate solution(s) $\theta_c \subset N(\theta_k)$ from the neighbourhood
  - Accept the candidate(s) and set $\theta_{k+1} = \theta_c$ (the new current solution) or reject it and set $\theta_{k+1} = \theta_k$
  - Increment $k = k + 1$

There are many factors contributing to the success of these methods. A few worth highlighting are the fact that they replicate processes that work well in nature, they have an applicability that is quite general, they are flexible when it comes to taking into account specific constraints in real cases and the compromise between solution quality and ease of implementation or computing time is not too severe [83].

As already mentioned, metaheuristics attempt to move from one solution to another better solution, or one set of solutions to another set of solutions and thus iteratively improving the solution quality until the method terminates. However, when simulation is used to estimate the performance, determining what constitutes a better solution becomes an issue. That is, given two solutions, θ_1 and θ_2 and simulation performance estimates $J(\theta_1)$ and $J(\theta_2)$, the fact that $J(\theta_1) < J(\theta_2)$ does not guarantee that $J(\theta_1) < J(\theta_2)$ every time. One now needs to look at statistical significance and see whether $J(\theta_1)$ is statistically significantly smaller than $J(\theta_2)$ [77]. This in itself would make traditional optimisation methods fail. Metaheuristic optimisation methods, however, overcome the challenges of simulation optimisation by making use of adaptive memory techniques and population sampling methods which allow the search to be conducted on a wide area of the solution space without getting stuck in local optima [14].

Six methods are now explored. Table 2.1 introduces the various heuristic methods that are to be discussed and what 'types', based on the above definitions as stated in [98], of methods these are.

Methods	Type of method
Genetic Algorithm	Local improvement
Simulated Annealing	Local improvement
Tabu Search	Local improvement
Non-linear Simplex Algorithm	Problem decompositioning/partitioning
Scatter Search	Randomly generated solutions method that reduces the solution space local improvement
Nested Partitions	Problem decompositioning/partitioning method that reduces the solution space local improvement

Table 2.1: Heuristic methods to be discussed in the literature review

2.2.1 Genetic Algorithm

Genetic Algorithms (GA) are related to the concept of evolutionary search in that they are described as a mechanism that mimics the genetic evolution of a species [85]. The basic concepts of GAs can be described as follows: a population of strings (which represent a possible combination of values which the variable or variables of interest can take on) is used and these strings are referred to as chromosomes in the GA literature. The recombination of these strings is then carried out using simple analogies of genetic crossover and mutation. This search is guided through the results of evaluating the objective function for each string in the population and based on this evaluation, strings that have higher fitness (those that represent better solutions) are identified and they are given more opportunity to breed [85]. This technique has been used in a wide variety of areas; a few examples are scheduling, timetabling and packing problems.

One of the GAs distinctive features is the use of string representations for each variable instead of the variable values themselves. The most common string representation is a binary representation although there are other string formations possible such as using numbers, trees, arrays or lists [85]. Although a distinctive feature, one notes that the use of strings can sometime be a disadvantage due to the fact that it requires extra computational time to recode these variables and there are also added difficulties if the variables being dealt with are discrete as opposed to continuous [85]. Examples of encoding other than bit encoding is octal encoding (which uses numbers 0-7) and hexadecimal encoding (which uses the numbers 0-9 and letters A-F). In permutation encoding (which is only useful for ordering problems), every chromosome is a string of numbers which represent a permutation vector where the permutation encoding represents a rank, an order, or a sequence number.

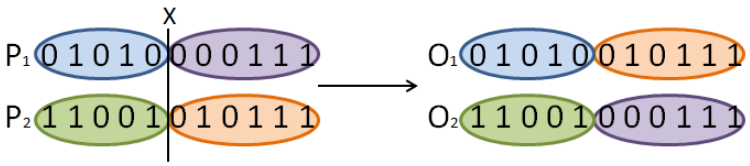

Figure 2.3: Example of a crossover operation used in genetic algorithms

GAs continue the biological analogy by calling the individual symbols in a particular string genes and the symbols of the alphabet, or number system, from which the genes are drawn alleles. Crossovers are simply a matter of replacing some of the genes in one parent by the corresponding genes of the other. There are many types of crossovers available. The most well-known crossover is one-point crossover. This is described as having two parents P_1 and P_2, the crossover point is selected, say point X for example. Two offspring, O_1 and O_2, are then created whereby O_1 is made up of the genes to the left of X of P_1 and of the genes to the right of X of P_2 and vice versa for O_2. An example of this is shown in Figure 2.3. One-point crossover can be extended to many point crossovers where r crossover points are chosen randomly. Many applications of GAs assume that crossovers are carried out stochastically where the probability of applying a crossover to any particular pair of strings is a value of $P_c < 1$ where P_c is the probability of crossover [85]. Other crossover mechanisms available are uniform crossover, three parent crossover, shuffle crossover precedence preservative crossover, ordered corssover and partially matched crossover all of which are explained in more detail in [100]. [103] displays a survey of the literature of the various merits of the different crossover mechanisms is given.

Mutation occurs when a gene is chosen at random and the allele value of that gene is changed. For example, in binary strings this just means complementing the bit value. There exists P_m, the mutation rate, which defines the probability with which a gene gets mutated and is used to decide how often a gene should be mutated [85]. An advantage of having mutations is that it prevents losing unexpected valuable genetic information in the population during selection and crossover operation [8].

GAs deal with populations of solutions as opposed to single solutions. This can be seen as a slight disadvantage in the sense that the length of the execution time may be long arising from the fact that there is a need to evaluate whole populations of candidate solutions [34]. However, the advantage arising from the use of populations of solutions is that it allows an increased search space exploration.

GAs have many advantages. The major ones are listed below:

- They are able to tolerate noisy, discontinuous and time varying function evaluations as opposed to many conventional optimising techniques which rely on unrealistic assumptions of linearity, convexity and differentiability. The only requirement in GAs is the ability to calculate some measure of performance (which, in itself however, can be highly complicated and non-linear) [85].

- GAs allow for a large number of different implementation choices [85].
- Many researchers have observed empirically that, given the right encoding and operators, GAs appear to be inherently robust. Although it is possible to fine-tune a GA to work better on a given problem, it is true that a wide range of parameter settings will give very acceptable results [85].
- GAs are very easy to modify. For some heuristics, slight changes in the problem can cause grave difficulties in the running of the heuristic. For GAs however, it is easy to change their structure in a way that models the variation in the original problem [85]. It is noted that a specific coding may have a significant impact on the GAs ability to find good solutions unless the operator used to search the space is carefully selected with respect to the coding [85]. Therefore, care must be taken to make sure that the GA structure suits the problem at hand.
- GAs have the ability to use the accumulative information about the initial unknown search space in order to move the next searches into useful spaces [8].
- Due to the fact that GAs work with a population of possible answers, they do not require initial estimates of the fitting parameters but require only the allowable range of each parameter [8].

Possible modifications of GAs are:

- The choice of population size - This should depend on the length of the strings defining the configurations. Some researchers use small population sizes in order to afford more repetitions of the algorithm (running the risk of under covering the solution space and failing to find a near optimal solution). Others prefer to repeat the algorithm fewer times in order to use large population sizes. Neither choice universally dominates in all optimisation settings. It appears necessary to spend more effort in tuning the parameters and making many runs to see what works best in any given problem [26].
- The string length - If the number of chromosomes is too low the possibility of movement operation by the GA will also be low and it only searches a small part of the search space [8]. However, if the number of chromosomes is too large this can result in an increased cost in terms of the time spent.
- The choice of the probability of crossover, P_c, and the probability of mutation, P_m, - The literature is clear on the best selection of P_m and P_c. Almost all researchers agree that the probability of crossover must be fairly high (above 0.3) while the probability of mutation must be quite small (less than 0.1) [26].
- The fitness calculation - One needs to try and avoid duplicates in order to avoid having to compute the same fitness repeatedly and distorting the process of selection [26].

- The stopping criteria - Some possible options are that there are only a fixed number of generations created or that there is a convergence in probability.
- The coding of configurations.
- The crossover operators.

As previously mentioned, a solution is selected for crossover based on fitness. Fit solutions will reoccur either unchanged or as part of their offspring whereas unfit solutions will not survive to the next generation. However, in a simulation context, the fitness of these solutions is subject to the same simulation noise and thus good solutions may be judged unfit prematurely. This leads on to the issue of how to account for simulation noise. One approach may be to account for the simulation noise in every generation by making sure that the selection for survival or crossover is done at a statistically significant level or by employing statistical screening procedures to select a subset of solutions for survival or crossover. However, GAs seem quite robust to inaccuracies in the performance estimates as it is actually preferable to have some randomness in the selection of solutions. This is an area that warrants further research [77].

GAs are not primarily designed for function optimisation but as models of efficient adaptive behaviour. Therefore, theoretical results are not concerned with convergence to a global optimum but with optimal or near-optimal sequences of decisions in the context of an unknown and uncertain environment [83]. The GAs have shown to be successful when the model is stochastic in nature [81].

A good example of where the GA has been implemented along with Monte Carlo simulation was performed in [70] where the approach is applied within the context of plant logistic management for what concerns the choice of maintenance and repair strategies.

2.2.2 Simulated Annealing

Simulated Annealing (SA) is another commonly used heuristic designed to permit escaping from local optima by allowing worsening moves which facilitates in eventually finding the neighbourhood of the true minimum. Due to its simplicity and versatility, SA has the distinction of being one of the most widely used techniques for function and combinatorial optimisation problems and has been proven successful in many applications [26, 104].

The method is inspired from the thermodynamic process of cooling (annealing) of molten metals to attain the lowest free energy state. When molten metal is cooled slowly enough it tends to solidify in a structure on minimum energy. This annealing process is mimicked by a search strategy [104].

The SA algorithm does not search for the best solution in the neighbourhood, $N(\theta_n)$, of the current solution, θ_n, but rather randomly draws a solution, θ, in $N(\theta_n)$. Then, if θ performs better than θ_n, in that $J(\theta) \leq J(\theta_n)$, then θ becomes the next current solution. If θ performs worse than θ_n then θ becomes the next current solution with probability $p(n)$ or θ_n remains the current solution with the complementary probability $1 - p(n)$.

The most common probability function used is the Boltzmann probability defined as

$$P(J(\theta), J(\theta^n), T) = \begin{cases} 1 & \text{if } J(\theta^n) - J(\theta) < 0 \\ exp(-\frac{J(\theta^n)-J(\theta)}{T \times K_B}) & \text{if } J(\theta^n) - J(\theta) \geq 0 \end{cases}$$

Where $J(\theta)$ is the current solution, $J(\theta^n)$ is the second solution, T is the temperature (which is decreased form iteration to iteration to model the thermodynamic process of cooling) and K_B is the Boltzmann constant (defined as 1.381). This probability function is most often used as it is the one which most closely resembles the physical process of cooling molten metals on which SA is based [61]. At the start of the algorithm any move is accepted. This is important as it allows the whole solution space to be explored. The temperature is then reduced from iteration to iteration and as such the acceptance probability decreases causing fewer new solutions from being selected as the current solution [104]. At high temperatures, the Boltzmann probability is close to one, so uphill steps (increasing energy) are frequently accepted allowing the system to climb up out of its local valley. At lower temperatures, the Boltzmann probability drops toward zero causing frequent rejection of uphill steps. Thus the system is prevented from escaping its local region and is progressively driven down in energy through the decrease of the temperature value.

Due to the fact that SA is a randomized algorithm it is recommended, especially when using SA in a simulation context, to make a large number of independent runs and look for good performance on average [83]. Also, in general it is noted that the running time which is necessary for an effective implementation of SA is usually long compared to the time used by deterministic heuristics, which is one of the disadvantages of SA [83].

In order for the SA algorithm to be effectively implemented the following control parameters need to be decided upon before the algorithm is implemented:

- *The acceptance probability* - As previously mentioned, the Boltzmann function is the most commonly known but there are many other functions available.
- *The cooling schedule* - These are non-increasing functions of time and are designed in such a way as to exclude almost all bad moves in the end.
- *The stopping rule* - There are many available stopping rules. Examples are that if a solution is not improved upon by a certain percentage from one iteration to the next then the algorithm is stopped or if the number of accepted moves was less than a specific percentage of the number of iterations then the algorithm is stopped.
- *The neighbourhood structure* - It has been seen that it is good to have a symmetric neighbourhood structure as it allows calculations in the neighbourhood to be carried out quickly and efficiently [26].
- *The number of iterations* - There are many ways in which this can be selected such as based on feedback from an iteration or based on the actual sample size or neighbourhood size [26].

These parameters can all be modified to suit the particular optimisation problem.

[46] gives necessary and sufficient conditions on the cooling schedule that guarantee almost sure convergence to an optimal solution in a deterministic setting . The main implication of these conditions is that the decrease in temperature must be logarithmic in time, however, in most implementations it is exponential. It has been shown that using logarithmic schedules in practice results in extremely long computing times [83]. The choice of a cooling schedule is a much discussed issue as there is a conflict between theory and practice [83].

In the SA algorithm the cooling schedule is fully defined by an initial temperature, a schedule for reducing/changing the temperature and a stopping criterion [48]. With this being said, cooling schedules can be grouped into two classes. The first is static schedules which must be completely specified before the algorithm begins while the second is adaptive schedules which adjust the temperature's rate of decrease from information obtained during the algorithm's execution [48]. Cooling schedules are almost always heuristic in the sense that they seek to balance moderate execution time with SA's dependence on asymptotic behaviour [48].

Many different cooling systems exist. Below are examples of the rules used for updating the temperature of some of the most commonly used cooling schedules. T_k represents the temperature at iteration k.

- Geometric: $T_{k+1} = \alpha T_k$ where $0 < \alpha < 1$ and is fixed throughout the algorithm
- Lundy: $T_{k+1} = \frac{T_k}{1+\beta T_k}$ where $0 < \beta < 1$ and is fixed throughout the algorithm
- Logarithmic $T_{k+1} = \frac{c}{log(k+1)}$ where c is a constant fixed throughout the algorithm
- Constant: The temperature is found experimentally by running a fixed temperature schedule for a range of temperatures and choosing the temperature that gives the best performance.

Many researchers have noted that SA can perform better if used in combination with other optimisation techniques. The most common is the use of a pre-processing heuristic to determine a good starting configuration for SA [104].

An advantage of the SA algorithm is that it does not require any functional derivative information and as a result it is not affected by discontinuous functions or non-linear functions. Another advantage is that it is a relatively easy algorithm to implement[34]. It has also been shown that the SA algorithm is an efficient approach when the choice of the initial design parameters is difficult or where hill-climbing methods fail or are unable to locate solutions other than the local ones [34].

Other types/extensions of simulated annealing are [112]:

- Multiple objective simulated annealing which uses a weighting scheme varied over multiple runs to generate an approximation of the set of non-dominated solutions.
- Pareto simulated annealing which uses annealing on a set of current solutions to simultaneously determine the non-dominated frontier.

[4] provides a modification of the SA algorithm which is designed for solving discrete stochastic optimisation problems. This modified algorithm also has hill climbing features which help prevent the algorithm from converging to local optima but differs from the traditional SA algorithm in the sense that it uses a constant temperature. They consider two approaches for estimating the optimal solution. The first approach uses the number of visits the algorithm makes to the different states (divided by a normalizer) to estimate the optimal solution. The second approach uses the state that has the best average estimated objective function value as the estimate of the optimal solution. Both of these approaches can be used for solving discrete optimisation problems when the objective function values are estimated using either transient or steady-state simulation. The authors also present a numerical example in which they perform their two approaches and show that the performance of their methods depends on the choice of a number of parameters, including the temperature, the neighbourhood structure and the number of estimated objective function values obtained in the different iterations [4]. [2] presents a new iterative method that combines the SA method and the ranking and selection procedure for solving discrete stochastic optimisation problems. Another example where SA has been used in the area of simulation optimisation can be seen in [19] where this technique is used for process scheduling.

2.2.3 Tabu Search

Broadly described, Tabu Search (TS) is a local search technique initiated by Fred Glover that is guided by the use of adaptive or flexible memory structures. More or less refined versions of the method have been used to 'solve' a large variety of combinatorial optimisation problems. A few examples are scheduling, vehicle routing, quadratic assignment, clustering and grouping, electronic circuit design, graph coloring and the traveling salesman problem.

[98] defines TS as a method that begins with a complete, feasible solution and continues developing additional complete solutions from a sequence of neighbourhoods. In order to escape from a local optimum, moves to neighbours with inferior solutions are permitted. In addition, a mechanism is used that prevents cycling back to recently visited solutions (in particular, the local optima). At the same time, recent solutions (or attributes of solutions) are maintained on a so called 'tabu' list preventing certain solutions from reoccurring for a certain number of iterations, called the size (or length) of the list. A record is kept of the best solution to date and its objective function value and the process is continued until the relevant stopping criteria have been met. The main characteristic that ensures the correct performance of TS is the way in which the solutions are selected from the neighbourhood [77].

It is noted that when additional complete solutions from a sequence of neighbourhoods are developed, the term neighbours has a broader meaning that the usual context of neighbourhood search. [37] defines this as in addition to considering solutions that are adjacent or close to elite solutions by means of standard move mechanisms, intensification strategies generate 'neighbours' by either grafting together components of good solution or by using modified evaluation strategies that favor the introduction of such components into a current (evolving) solution.

The diversification stage on the other hand encourages the search process to examine unvisited regions and to generate solutions that differ in various significant ways from those seen before.

A possible modification to the TS algorithm is to include aspiration criteria. The purpose of these aspiration criteria is to override a solution's tabu state which leads the solution to be included in the allowed solution.

TS is an adaptive method which can be used in conjunction with many other methods (such as linear programming algorithms) and specialised heuristics which it directs to overcome their limitations of getting stuck at local optima. Research and computational comparisons have shown that TS has the ability to obtain high quality solutions with little computational effort and also has been shown to often dominate alternative methods tested [18].

Before the TS algorithm is implemented the characteristics below need to be defined. The choice of these aspects can be done in such a way that best suits the type of problem at hand.

- The choice of the move attributes that need to be recorded in the tabu list [83].
- The choice of the aspiration criteria [83].
- The selection of the stopping rule [83].
- The specification of the neighbourhood structure [83].
- The neighbourhood sizes and candidate lists - When a problem has many dimensions it may be very computationally expensive to examine the complete neighbourhood and as a result it may be better to examine only some of the regions of the neighbourhood that contain desireable features. There are many ways in which these desirable regions can be selected. An example is to use a neighbourhood decomposition strategy which breaks down the neighbourhood at each iteration into coordinated subsets and uses aspiration criteria (as an example) of linking how subsets are examined in order to visit less appealing regions less often [26].
- The length of the tabu list - If the tabu list size is too small then cycling may occur and if the list is too large appealing moves may be forbidden.
- The type of the tabu list - There is the choice that the tabu list may be either static or dynamic. Research has shown that the advantage of better solution quality in dynamic tabu lists exceeds the disadvantage of the costs of implementing the greater flexibility.

There are also added features to TS that can be used to modify the algorithm to suit the problem. These are including a penalty function that comes into play when constraints are violated, including a probability when choosing from a selection of candidate solutions and implementing a frequency based memory whereby both short and long term memory are used simultaneously [98].

The noisy estimates of simulation optimisation affect many parts of TS. Firstly, simulation noise is relevant when in every step the best non-tabu solution is selected from the neighbourhood of the current solution which raises the issue of how accurately this solution should be

selected. An area where further research is needed is in deciding if it is worthwhile to spend considerable simulation effort on each solution in the non-tabu list to make the correct selection with a high probability or is this simulation effort better spent by making a quick selection and exploring more of the search space [77].

Tabu lists can be more restrictive than necessary as they exclude many more solutions that those met in the last iteration but at the same time, tabu lists consisting of move characteristics can still have cycling present even if the list is unbounded [83]. The main aim of the tabu list, however, is to prevent this cycling from occurring. There is the problem that when simulation noise is present each time a solution is visited a different neighbourhood solution may be selected, even if the tabu list is identical. Also, if an incorrect solution is selected and becomes tabu it might be beneficial to revisit the solution to fix the mistake but this will not be possible due to the fact that the solution is now tabu. The cycle can however be broken without the tabu list due to the simulation noise and allowing the algorithm to revert back to an old solution enables the search to correct mistakes made due to the simulation noise [83].

Advantages of tabu search are that it performs very well when only short term memory strategies are used and can perform considerably better if additional intensification and diversification strategies are used. There is also the advantage that tabu search can be enhanced with several additional strategies. A disadvantage, however, is that although it often achieves very good performance it may require time due to the intensive fine tuning that will be required [27].

[63] proposes alternative strategies to perform simulation within a simulation optimisation algorithm based on TS. They test these strategies empirically on the stochastic knapsack problem and their results have shown that the way simulation is implemented and the number of simulation replications have a large effect on the performance of tabu search.

An example where TS have been used in a simulation optimisation context can be seen in [22] where TS is used along with a simulation model of a Just-In-Time (JIT) system to find the optimum number of kanbans.

2.2.4 Simplex Search

The Nelder-Mead method was originally developed for deterministic optimisation and then later evolved and developed into the Nelder-Mead simplex search method for stochastic optimisation [13]. This method has been used in a wide number of unconstrained optimisation scenarios [25]. A few areas where this method has been applied are are that of physics, crystallography, biology, chemistry and health care [25]. To prevent confusion, it is noted that the non-linear simplex algorithm is not directly related to the popular simplex method for linear programming [102].

A simplex is a geometric object that is the convex hull of $p+1$ points not lying in the same hyperplane. The convex hull is defined as the smallest set enclosing the $p+1$ points such that the line segment connecting any two points in the set lies in the set [102].

Basically, the simplex method starts with a search that evaluates the points in a simplex consisting of $p+1$ vertices (where p is the number of parameters) in the feasible region. It

proceeds by continuously dropping the worst points in the simplex and adding new points determined by the reflection of this worst point through the centroid of the remaining vertices [9]. The method uses four basic procedures: reflection, expansion, contraction and shrinkage. Through these procedures the simplex can successfully improve itself and get closer to the optimum by moving the simplex in a direction in which the objective function is improved upon [25]. The process is terminated once the simplex is sufficiently small and the solution is taken as a point inside this simplex [102].

[102] outlines the Nelder-Mead Simplex Algorithm as follows for minimisation problems:

Step 0 *Initialization* - An initial set of $p+1$ extreme points in the feasible region, θ_i for $i = 1, 2, ..., p+1$, is generated. These extreme points represent the vertices of the initial simplex. The objective function, $J(\theta_i)$, is then calculated for each extreme point. Lastly, the values for the algorithm coefficients, α, β, γ and δ are set.

Step 1 *Reflection* - The vertices where the maximum, second highest and minimum objective function value are then found and these points are represented as $\theta_{\max}, \theta_{2\max}$ and $\theta_{\min}$ respectively. Define θ_{cent} to represent the centroid (mean) of all θ_i except for $\theta_{\max}$. Then a new candidate vertex θ_{refl} is generated by reflecting $\theta_{\max}$ through θ_{cent} according to the function $\theta_{\text{refl}} = (1+\alpha)\theta_{\text{cent}} - \alpha\theta_{\max}$ where $\alpha > 0$.

Step 2a *Accept reflection* - If $J(\theta_{\min}) \leq J(\theta_{\text{refl}}) < J(\theta_{2\max})$, then θ_{refl} replaces $\theta_{\max}$ in the simplex and proceed to step 3 otherwise go to step 2b.

Step 2b *Expansion* - If $J(\theta_{\text{refl}}) < J(\theta_{\min})$ then the reflection is expanded according to $\theta_{\text{exp}} = \gamma\theta_{\text{refl}} + (1-\gamma)\theta_{\text{cent}}$ where $\gamma > 1$ otherwise go to step 2c. If $J(\theta_{\text{exp}}) < J(\theta_{\text{refl}})$ then θ_{exp} replaces $\theta_{\max}$ in the simplex otherwise the expansion is rejected and θ_{refl} replaces $\theta_{\max}$. Go to step 3.

Step 2c *Contraction* - If $J(\theta_{\text{refl}}) \geq J(\theta_{2\max})$, then the simplex contracts to reflect the poor θ_{refl}. Consider two cases:

(i) $J(\theta_{2\max}) \leq J(\theta_{\text{refl}}) < J(\theta_{\max})$, sometimes called outside contraction, or

(ii) $J(\theta_{\text{refl}}) \geq J(\theta_{\max})$, sometimes called inside contraction.

The contraction point is determined by $\theta_{\text{cont}} = \beta\theta_{\max/\text{refl}} + (1-\beta)\theta_{cent}$ where $0 \leq \beta \leq 1$ and $\theta_{\max/\text{refl}} = \theta_{\text{refl}}$ in case (i) or $\theta_{\max/\text{refl}} = \theta_{max}$ in case (ii). In case (i), if $J(\theta_{\text{cont}}) \leq J(\theta_{\text{refl}})$, the contraction is accepted. In case (ii), if $J(\theta_{\text{cont}}) < J(\theta_{\max})$, the contraction is accepted. If the contraction is accepted replace $\theta_{\max}$ with θ_{cont} and go to step 3. If the contraction is not accepted then go to step 2d.

Step 2d *Shrink* - The contraction has failed and the entire simplex shrinks according to a factor $0 < \delta < 1$, retaining only $\theta_{\min}$. This is done by replacing each vertex θ_i by $\delta\theta_i + (1-\delta)\theta_{\min}$. Go to step 3.

Step 3 *Termination* - Stop if a convergence criteria is satisfied or if the maximum number of function evaluations has been reached otherwise return to step 1.

This method can be extended to the complex search method which is just a constrained form of the simplex search. The search procedure is similar to the simplex method except that additional effort is made to prevent the simplex from leaving the feasible region [18].

Advantages of the method are that it is easy to use, computationally efficient and does not need the derivatives of the function under evaluation [25]. Also, due to the fact that the method is based solely on the ranking of the responses, it can make progress in the presence of relatively small random perturbations since small perturbations do not change the ranks of the function values at the simplex points [13].

The rescaling operations allow the simplex to adapt to a function's local behaviour. Unfortunately these operations can lead to the method terminating too quickly when differences in function values across the simplex are dominated by the stochastic perturbations. Inappropriate termination can lead to substantial error in the estimated optimum for response functions with large stochastic components.

Disadvantages are that the method is not guaranteed to reach the global optimum and the method is very sensitive to the choice of the initial points (this can be overcome through the use of hybridisation which is explained in more detail in [25]). Also, there is the implementing problem involving the handling of feasible constraints and also the fact that the assumption of a convex feasible region is made [18]. Since the Nelder-Mead simplex search is a local search method, no guarantee is given for finding the global optimum. Therefore, when optimising a stochastic objective function, multi-start using multiple starting points and/or multiple searches from the same starting point should be considered as this might lead to a more accurate solution.

[13] gives analytical and empirical results describing the performance of the Nelder-Mead Simplex Search when it is applied to a response function that incorporates an additive white-noise error and they go onto using the results obtained to develop new modifications of the Nelder-Mead Simplex Search to yield improved estimates of the optimal expected response. These modifications include that the best point in the simplex is reevaluated at each shrink step and that the simplex is reduced by around 10% at each shrink step. More adaptations of the Nelder-Mead simplex method for optimisation of simualtion models can be seen in [75]. The adaptations are shown on a test-case small microsimulation model.

2.2.5 Scatter Search

Scatter search is another metaheuristic related to the concept of evolutionary search. [35] defines scatter search as "an information driven approach, exploiting knowledge derived from the search space, high-quality solutions found within the space, and trajectories through the space over time." They go on to say that the combination of these factors is what creates a highly effective solution process. Scatter search has been used effectively in the following areas to name a few: vehicle routing, quadratic assignment, crew scheduling and graph drawing [32].

In every step a scatter search algorithm considers a set $\theta_k \in \Theta$ of solutions called the

reference set. Similar to the genetic algorithm approach, these solutions are then combined into a new set $\theta_{k+1} \subset \Theta$. This is generally achieved through the linear combinations of the solutions in the reference set which thus defines the neighbourhood [77]. Unlike genetic algorithms, the reference set in scatter search algorithms typically have 20 solutions or less [36]. This is due to the fact that diversification can be achieved through the way in which the solutions in the reference set are combined [36].

[36] defines the basic template for implementing scatter search as follows:

- *A Diversification Generation Method* which generates a collection of trial solutions that are diverse. This is generally done using an arbitrary trial solution (or seed solution) as an input [32]. The reference set that is generated does not need to contain the best solutions only but can also contain solutions that help improve the diversity of the set even if their associated objective function value is not as good as other solutions.

- *An Improvement Method* to transform a trial solution into one or more enhanced trial solutions. (For this to happen, neither the input nor the output solutions are required to be feasible but it is often expected that the output solutions are in fact feasible. If no improvement of the input trial solution results, the "enhanced" solution is considered to be the same as the input solution.)

- *A Reference Set Update Method* to build and maintain a reference set consisting of the b 'best' solutions found, organized to provide efficient accessing by other parts of the method. Solutions enter the reference set depending on their quality or on their diversity.

- *A Subset Generation Method* to operate on the reference set, to produce a subset of its solutions as a basis for creating combined solutions. This method is not typically limited to combining just two solutions but more generally consists of creating subsets of different sizes [71]. An example of an efficient approach is outlined below. This approach selects representative subsets of different sizes by creating subset types defined as [71]:

  - Subset type 1: all 2-element subsets
  - Subset type 2: 3-element subsets derived from the 2-element subsets by augmenting each 2-element subset to include the best solution not in this subset.
  - Subset type 3: 4-element subsets derived from the 3-element subsets by augmenting each 3-element subset to include the best solutions not in this subset.
  - Subset type 4: the subsets consisting of the best i elements, for $i = 5$ to b.

- *A Solution Combination Method* to transform a given subset of solutions produced by the Subset Generation Method into one or more combined solution vectors.

According to [71] the main advantages of Scatter Search are that it can help solve a wide variety of optimisation problems from both classical and real world settings, it provides unifying principles for joining solutions based on generalized path constructions in Euclidean space and by utilizing strategic designs where other approaches tend to randomize and lastly, the ability

to use intensification (focusing on examining neighbours of elite solutions) and diversification methods (focusing on examining unvisited regions) to enhance the method [71].

An example where the scatter search technique has been implemented in a simulation optimisation context can be seen in [35].

2.2.6 Nested Partitions

Nested Partitions (NP) is a randomised optimisation technique aimed at solving global optimisation problems developed by Shi and Ólafsson [97]. Originally, this method was developed for finite feasible regions but can be extended to problems where the feasible region is either countably infinite or uncountable and bounded [97]. NP is a global optimisation technique due to the fact that the method always uses the entire set of feasible solutions. This means that given a current set $\theta_k \subseteq \Theta$ of solutions, the neighbourhood is always $N(\theta_k) = \Theta$.

[97] describes the method as follows; whenever the algorithm is iterated it is assumed that there is a region (a subset of the feasible space) that is considered to be the most promising. This region is then itself further partitioned into M regions and the surrounding area is aggregated into one region so that, in total, there are $M + 1$ disjoint regions which together make up the feasible region. Each one of these regions is then sampled using some random sampling scheme and a promising index is given to the region based on the performance of the objective function with the given sample. This index is then used to decide which region becomes the next most promising region. If one of the sub-regions becomes the most promising then the algorithm continues as normal with this sub-region becoming the new most promising region. If the surrounding region is found to be the most promising then the algorithm can either backtrack to a region that contains the old most promising region or the algorithm can start again with the whole feasible region. This process continues until the stopping criteria have been met.

The defining characteristic of this method is the adaptive distribution that is used to obtain sample solutions from this neighbourhood. As this method progresses through a sequence of set partitions, with each partition nested within the last, the sampling focuses on sets that appear promising or, in other words, sets where the optimal solution is believed to be contained. Once the random candidate solutions have been generated according to this adaptive distribution, they are always accepted, and the search continues with a new sampling distribution [97].

The NP method includes a built-in mechanism for recovering from incorrect moves. Incorrect moves may arise due to the presence of simulation noise. In every step, the neighbourhood of the current solution is the whole solution space, so any solution has a positive probability of being selected to be evaluated. This includes the surrounding region, that is, those solutions that are not in the set currently considered most promising. If the best solution is found in the surrounding region, the algorithm backtracks to an earlier (larger) set of solutions. This can be viewed as an automated mechanism to recover from an incorrect move made either due to the use of random samples to guide the search, or due to the simulation noise. These global search and backtracking mechanisms of the NP method can therefore be used as a global guidance system to local improvement methods in simulation optimisation [77].

The main approach behind the NP method is systematically partitioning the feasible region into sub-regions, evaluating the potential of each region, and then focusing the computational effort to the most promising region. This process is carried out iteratively with each partition nested within the last. The computational effectiveness of the NP method relies heavily on the partitioning, which, if carried out in a manner such that good solutions are clustered together, can reach a near optimal solution very quickly [77].

In general, the NP method favours solutions that have many good solutions and no bad solutions rather than a region that has one good solution and many bad ones.

The effect of the simulation noise comes into play in the NP method in the selection of a region as the most promising region. If the performance estimates for each sample solution are noisy, the next most promising region is selected with less confidence. However, this is already a noisy selection even for deterministic problem since there are two sources of randomness. These come from:

- The sampling error due to a small sample of solutions being used to estimate the overall region which comprises of a large set of possible solutions.
- The noise that results from the fact that for each of the sample solutions, their performance is estimated using simulation.

Special structure, local search, any heuristic search and mathematical programming can all be incorporated into the NP framework to develop optimisation algorithms that are more effective in solving large-scale optimisation problems than when these methods are used alone.

It is noted that before the NP algorithm is initiated the number of partitions created, the sampling scheme as well as the stopping criteria need to be decided upon

The major advantages of the NP method are taken from [77] and [96] and are listed below:

- It is a generic optimisation technique capable of supporting different partitioning techniques, sampling methods, promising indices and backtracking rules.
- One does not necessarily have to have a good starting point like in simulated annealing where the starting point can have a significant impact on the outcome. The NP method takes a global view in each iteration and therefore the computational effectiveness of the NP method relies heavily on the partitioning which, if carried out in a manner such that the good solutions are clustered together, can reach a near optimal solution very quickly.
- The algorithm always spends the most computational effort in the region that is considered the most promising at any given iteration which makes the solutions found by the method insensitive to small deviations from the best solution found.
- The NP method has been found to be very effective at solving objective functions that are either known analytically or are noisy or even when they must be evaluated using an external process.

- In the NP method, few assumptions are made about the structure of the problem. Instead the partitioning strategy implicitly imposes a structure on the problem.
- The NP method has the strength that it can take advantage of parallel computer structures.

The NP method can be varied or grouped in different ways as defined in [97] and listed below:

- There are two main method types in the NP method. The first is a branching method that divides the solution space into partitions called branches and focuses the computational effort on obtaining tight bounds for each branch. The second is decomposition methods whereby the problem is 'solved' by either eliminating constraints or variables. For example, this method is helpful in data mining problems where it may be necessary to evaluate the performance with only a fraction of the data since the amount of data is often too large to work with all at once.
- There are also pure NP methods where the feasible solutions are generated using simple uniform random sampling and there are knowledge based NP methods that use knowledge about the problem at hand to generate feasible solutions.

Examples where the NP method has been used along with other methods to solve a simulation optimisation are given in [94] and [95]. In [94] a new algorithm to tackle stochastic discrete resource allocation problems is proposed. The algorithm that is used combines the NP method, ordinal optimisation and an efficient simulation control technique. The resulting hybrid algorithm retains the global perspective of the NP method and the fast convergence properties of the Ordinal Optimisation and numerical results demonstrate that the hybrid algorithm can be effectively used for many large-scale stochastic discrete optimisation problems. In [95], another algorithm is used which integrates NP with an optimal computing budget allocation method. Again, the resulting hybrid algorithm retains the global perspective of the NP method and the efficient simultaneous simulation experiments of the optimal computing budget allocation and the numerical results demonstrate that the hybrid algorithm can be effectively used for a large-scale discrete resource allocation problem.

2.2.7 Remarks from Heuristics

The principle difference among heuristic methods is how and where domain specific knowledge is used. For example, in simulated annealing, such knowledge is mainly included in the cost function. Elements involved in a perturbation are selected randomly and perturbations are accepted or rejected accordingly based on the specified criterion which is a function of the cost. The cooling schedule has a major impact on the algorithm performance and particular problem instance. In the case of genetic algorithms, domain specific knowledge is exploited in all phases. The fitness of individual solutions incorporates domain specific knowledge. Selection for reproduction, the genetic operations as well as generation of new population also incorporate

a great deal of heuristic knowledge about the problem domain. Tabu search is different from the above heuristics in that is has an explicit memory component. At each iteration, the neighbourhood of the current solution is partially explored and a move is made to the best non-tabu solution in that neighbourhood. The neighbourhood function as well as the tabu list size are content and problem specific. The direction of the search is also influenced by memory structures [115].

Advantages of heuristic methods is that they constitute a broad class of optimisation techniques that can be applied to solve both deterministic and stochastic optimisation problems with either discrete or continuous decision parameters [6]. Also, efficiency of these methods can be achieved through the way in which a neighbourhood is searched as well as through the correct size selection of the neighbourhood [6]. When dealing with simulation optimisation, there are a few desirable features that these methods possess. These methods incorporate both global and local search components and manage to maintain the right balance between the two as the search progresses (although this does depend heavily on the choice of parameters for many of the methods) [6]. Other desirable features are almost sure global convergence, the ability to focus the search in desirable regions, the ability to move quickly within the search region and, lastly, the ability to focus the search in areas of the feasible region that either appear to be desirable (have relatively low objective function values in a minimising context) or have not been sampled much before [6]. It is noted that heuristic methods can also be referred to as random search methods in various literature.

2.2.8 A-Teams

An A-Team is a multi-population, multi-agent system for solving optimisation problems [18]. These are also sometimes known as asynchronous teams. A-Teams are suitable for multi-criteria simulation optimisation problems and therefore represent one of the fastest growing areas of simulation optimisation [18]. The A-Team framework enables different problem solving strategies to be easily combined, each in the form of an 'agent', and enables these 'agents' to cooperate to improve the quality and diversity of the resulting solution. A-Teams have different applications, such as, resolution of nonlinear systems of equations, optimisation problems, civil engineering, networking and power systems among other applications [12].

A-Teams can be outlined as a method which uses a team of agents to optimise a population of solutions. Agents are the modules from which problem-solving systems can be built. Structurally, an agent can be thought of as a bundle of sensors, decision-makers and actuators. Behaviorally, an agent can be thought of as a mapping from an in-space (all the things the agent can sense) to an out-space (all the things the agent can affect). These spaces may overlap [107]. (Note that by these definitions, a great many, very different things qualify as agents, including thermostats, computer programs, mobile robots, insects, people, societies and corporations.) These agents cooperate by sharing access to populations of candidate solutions. The purpose of an agent is either to create, modify or remove solutions for a population. As time progresses, the quality of the solutions in the population improves as better solutions are added to the

population and poorer solutions are removed from the population. The agents cooperate with one another as one agent works on a solution produced by another [12].

Each agent is however autonomous and asynchronous [12]. They can be broken down into three categories: constructors, improvers and destructors. The constructor agents create initial solutions and add them to the population. The improvers then select one or more existing solutions from the population and produce new solutions that are added to the population. It is noted that the improvers don't necessarily improve the solution as sometimes the modifications they make might result in worse performing solutions. Lastly, the destroyers keep the population to the specified size by removing badly performing solutions and keeping well performing solutions [12]. Agents can differ in many ways. For example, some may iterate quickly while others are slow and some may make small improvements while others make radical or even innovative changes to the candidate-solutions [107]. Examples of agents are humans acting as decision makers, the simulated annealing algorithm and the tabu search algorithm.

A-Teams have many advantages. The first is that the process is modular. This is due to the fact that the agents are autonomous and do not depend on each other for either execution or communication which means that in development, the agents can be created independently of each other. Also, agents can be easily added or deleted from the system at any time. Secondly, due to the way that the A-Teams are designed, it is suited to parallel implementation. Thirdly, A-Teams are robust in the sense that the failure of one agent will not result in all the other agents failing [12].

For a more detailed explanation of this method see [107].

2.3 Gradient Based Search Methods

Broadly, in gradient based search methods, the 'best guess' of the optimal parameter is updated iteratively based on an estimate of the gradient of the performance measure with respect to the parameter [5]. Using gradients when managing continuous valued control problems is very useful as it helps obtain improved solutions based on an iterative scheme and helps to keep the problem traceable. This idea is extended upon within a simulation optimisation context [32].

Gradient based search methods estimate the response function gradient in order to assess the shape of the objective function and use deterministic mathematical programming techniques [18]. The gradient estimation method presupposes that J is differentiable in θ, so in the simulation optimisation context, the setting where Θ is a continuous space is therefore considered [33]. There is much literature available on search methods developed for non-linear programming problems which these methods make use of [9]. These methods also have the advantage that they are not only useful in simulation optimisation settings but also in other areas such as sensitivity analysis [32]. One notes, however, that an evaluation of the gradient is needed to determine the appropriate direction of motion which limits the application of the gradient based search methods to problems where the gradient can be obtained accurately and easily [57].

A very key assumption in gradient estimation is that every estimate of the objective function

and of a particular parameter is expensive to generate. This means that great care goes into estimating the values properly and to make maximum use of the information that they contain [32]. As a result, the two most important features used in determining the success of these methods are the reliability (due to the fact that simulation responses are stochastic and a large error in gradient estimation may result in a movement in an entirely wrong direction) and the efficiency (due to the high costs associated with the simulation optimisation) [9].

Gradient based search methods can be broken down into indirect gradient estimators and direct gradient estimators. Indirect estimators have the advantage that they offer greater generality. Their two predominant characteristics are that, one, they only estimate an approximation of the true gradient value and, two, they use only function evaluations from the original system of interest. Direct gradient estimators, on the other hand, try to estimate the true gradient using some additional analysis of the underlying stochastics of the model. They also assume that the simulation output is coming out of a black box such that no knowledge of the underlying mechanics of the simulation model is used in deriving the estimators such as knowing the input probability distributions and no changes are made in the execution of the simulation model itself [32].

The different gradient based search methods in a simulation optimisation context are discussed below. These are finite difference estimation, infinitesimal perturbation analysis, frequency domain analysis, likelihood ratio estimation and stochastic approximation. It is noted that not much detail is presented on these methods because although they are relevant in the context of simulation optimisation, when implementing these methods the user needs to have thorough knowledge of the simulation model as these methods are very much problem dependent. Due to the fact that the optimisation techniques are to be applied to a simulation model that was created by an outside source and a complete understanding of the intricacies of how it works are not had, these techniques are not applied to this model.

2.3.1 Finite Difference Estimation

The simplest method of estimating the gradient is through the partial derivatives of the objective function which are calculated as

$$\frac{\partial J}{\partial \theta_i} = \frac{J(\theta_1, ..., \theta_i + \Delta\theta i, ..., \theta_p) - J(\theta_1, ..., \theta_p)}{\Delta\theta_i}$$

where J is the objective function and p is the number of parameters [9]. This means that in order to estimate the gradient at each point, $p + 1$ evaluations need to be carried out. Also, in order to add reliability one might require multiple observations for each derivative [9].

This method involves changing the value of the parameters one at a time while keeping the other values constant. One notes that if the changes of the parameters are kept quite small then the output can be extremely noisy because of the fact that the output is stochastic. There is therefore a choice to be made between bias and variance and therefore, unless the parameter vector is suitably standardised, one needs to choose the perturbations for each component separately [32].

A disadvantage of this method is that it takes a long time to carry out and it has a slow convergence rate [68]. Another disadvantage is that to increase the numerical accuracy of the derivative, both computed simulation points in the definition must be within a very small distance of each other. This closeness leads to difficulty of handling derivatives when the function is noisy which is often the case in simulation optimisation. The advantage, however, is that it is relatively straightforward to implement and the most generally accepted gradient based search method [5].

An example of this method used in conjunction with the Hookes and Jeeves pattern search technique is presented in [82].

2.3.2 Infinitesimal Perturbation Analysis

Infinitesimal perturbation analysis (IPA) method estimates all gradients of the objective function from a single simulation experiment. This method is based on the concept that through changing a decision parameter of a system by an extremely small amount, the sensitivity of the response of the system to that parameter can be estimated by tracing its pattern and spread through the system. This will be a function of the propagations that terminate before having a significant impact of the response of interest [9]. The basic idea of IPA is to generate a sample path ω, viewed as a sequence of $U(0,1)$ variates, and for a fixed ω, observe the effect of an inifitesimal perturbation on θ by propagating it over the sample path, assuming that the sequence of events do not change and that the events only 'slide' in time. The gradient estimation is taken as the gradient of the objective function for that fixed value of ω [67].

IPA is a well-defined technique which is easy to derive and implement with very little computational overhead. Intuitively, it is the estimator resulting from assuming that small changes cause no event order changes [5].

The advantages of this method is that it only requires one simulation run, it is highly efficient and easy to implement although it does have limited applicability. One notes that if IPA is not applicable, extensions are necessary and these extensions are usually not as straight forward [5].

A disadvantage of IPA is that certain conditions need to be met in order for this method to be implemented. For example, if a result of a perturbation of a given parameter (the sequence of events that govern the behaviour of the system) changes due to the random nature of the simulation, the results obtained from the perturbation analysis may not be reliable [9]. Other disadvantages are that the modeller needs to have a thorough knowledge of the simulation model in order to add additional tracking capabilities due to the fact that this method requires a modification of the simulation source code to incorporate the gradient computation during a simulation run [9]. Also, since all estimates come from one simulation run, it means that the estimators derived are often biased and inconsistent [18]. However, the fact that all the derivatives can be derived from one simulation run makes the method more efficient [9]. One notes that IPA cannot handle 'bad' performance measures such as indicator functions, non-smooth systems etc. [32].

There are other extensions to this method which include smoothed perturbation analysis

and IPA variants. For more information of the IPA method itself and its variants see [17] and for a list of examples where this technique has been implemented see [108].

2.3.3 Frequency Domain Analysis

The frequency domain analysis (FDA) method increases the amount of information that can be obtained from each simulation run by shifting the analysis into the frequency domain. A feature of simulated systems is that one has complete control over the input factors. FDA exploits this feature and was developed to be used as a tool to identify the most significant parameters of a simulation model. In a FDA experiment values for continuous factors are varied during a run according to sinusoidal oscillations. Different frequencies are assigned to each factor. If the simulation response is sensitive to changes in a particular factor, then oscillating that factor induces predictable oscillations in the response. Varying the values of unimportant facts does not alter the response. The spectrum of the simulation output measures the relative strength of these oscillations [93]. The gradients are estimated by analyzing the power spectrum of the simulation output function which is affected by inducing specific sinusoidal oscillations to the input parameters and from this the simulation model can be optimised [92].

This method provides significant savings over the traditional design of experiment approaches such as factorial design [5]. The output variable values are then subjected to spectral (Fourier) analysis i.e. are regressed against sinusoids at the input driving frequencies. If the output variable is sensitive to an input parameter, the sinusoidal oscillation of the parameter should induce corresponding (amplified) oscillations of the response [18]. FDA aims to address three questions [18]:

1. How does one determine the unit of experimental or oscillation index?
2. How does one select the driving frequencies?
3. How does one set the oscillation parameters?

The vectors of input parameters are modelled as follows

$$\theta(t) = \theta_0 + \alpha sin(\tilde{\omega} t)$$

Where θ_0 is the parameter vector of interest, α is the vector of oscillation amplitudes and $\tilde{\omega}$ is the vector of oscillation frequencies called the driving frequencies (which are assumed to be distinct in order to be able to discriminate between the contributions of each parameter) [5]. The time variable t is usually not the simulation time. In order to apply FDA the determination of an appropriate 'time' is a non-trivial problem called the indexing problem. More information on this problem can be found in [74]. By finding the values in the above equation one is then able to answer the three questions listed above [5].

Similarly to IPA, one notes that a large disadvantage in this method is that the modeller needs to have thorough knowledge of the simulation model in order to add additional tracking capabilities [9]. However, this method has the advantage that it will typically require fewer

computer runs than conventional run-orientated simulation experiments, several input factors can be studied in the same run and non-linear effects, such as products of integer powers of the input factors, can be detected with no additional experimentation [93].

An example of where FDA is used in a simulation optimisation context is in [93] where the frequency domain methodology was first introduced as a screening tool for continuous input factors in discrete-event simulations.

2.3.4 Likelihood Ratio Estimators

In likelihood ratio estimators, the gradient of the expected value of an output variable with respect to an input variable is expressed as the expected value of a function of input parameters and simulation parameters [18]. For this method, the estimates obtained become more and more reliable as the number of simulation runs increases [18]. It is noted, however, this method only requires one simulation run in order to be implemented [27]. It has been noted [38] that the construction of a likelihood ratio that has desirable computational and variability characteristics is an important issue in the development of likelihood ratio gradient estimators.

Advantages of this method are that it requires only model input parameters, it is easy to implement and often works where IPA does not [27, 5]. Also, from a single simulation, one can estimate the derivative of the performance measure together with the performance measure itself.

One major disadvantage of this method is the large variance associated with the estimates [68]. Also, one may encounter difficulties in handling parameters not in the distribution (the so-called 'structural' parameters) and therefore may require a change of variables [32]. Also, there is the issue that the resulting estimator may have variance problems for some systems [5].

[39] provides an overview of the likelihood ratio gradient estimator and shows how it can be applied to discrete event simulations of arbitrary complexity. It is also noted by Tekin and Sabuncuoglu [108] that likelihood estimator application usually finds application areas when it is used in conjunction with stochastic approximation (see below for a description of this method).

2.3.5 Stochastic Approximation

Another form of gradient based estimation is stochastic approximation. This method was first introduced by Robbins and Monro [87] and Keifer and Wolfowitz [60]. Stochastic approximation is an iterative update scheme on the parameter that takes the general form for finding a zero of the objective function gradient. The algorithm attempts to mimic the gradient search method in deterministic optimisation, but in a rigorous statistical manner taking into account the stochastic setting [27]. This method has been designed to be used for continuous variable problems although there has been work done on trying to apply it to discrete variable problems [27]. Much research for optimisation of discrete-event systems via simulation using stochastic approximation methods has been done [5].

Stochastic approximation can be defined as follows:

$$\theta_{n+1} = \pi_{theta}(\theta_n - a_n \hat{\nabla} J(\theta_n))$$

where $\hat{\nabla}$ denotes the estimate of the gradient, a_n is the gain sequence and π is a projection back into the feasible region once θ is taken out of the feasible region. The main factors in this method are obtaining a gradient estimate and choosing an observation length for each iteration [5].

An advantage of this method is that the optimum of the expected value of the response can be reached using noisy observations [9]. Disadvantages of this method are that a number of iterations of the recursive formula will be required to obtain the optimum. Also, one requires $p + 1$ observations for each iteration (where p is the number of parameters) [9]. The difficulty of incorporating the constraints into the optimisation also exists as well as the sensitivity of the early transient convergence rate to the choices of the sequences and the fact that it might be hard to project onto the feasible region. Lastly, there is the issue that the stochastic approximation algorithm converges to a local optimum but it is noted that this can become less of a disadvantage if 'noise' is introduced into the algorithm [32].

[89] presents an example of optimisation using stochastic approximation and Monte Carlo simulation. Their application is to harvesting the Atlantic Menhaden with the objective to maximise the expected utility of the random catch over a 25 year period.

2.4 Statistical Methods

2.4.1 Importance Sampling

The idea behind importance sampling is that certain values of the input random variables in a simulation have more impact on the parameter being estimated than others. If these 'important' values are emphasized by sampling more frequently, then the estimator variance can be reduced. Hence, the basic methodology in importance sampling is to choose a distribution which 'encourages' the important values [101].The main application of this method is to achieve significant speed ups in simulation involving rare events [18]. The importance sampling methods force a larger number of the rare events from the underlying distribution into a sample that is used for simulation [45]. The principle of importance sampling is to simulate the system under a different probability measure (e.g. with a different underlying probability distribution) so as to increase the probability of typical sample paths involving the rare event of interest. The distribution of the sample points is concentrated in the parts of the interval that are of most 'importance' instead of spreading them out evenly. This biasing of the PDF is compensated by properly weighting the observable outputs. The weights are dependent upon the input random variables. The PDF is biased in such a way that the variance in the importance sampling simulation is minimized. It is this reduced variance property of importance sampling which leads to large savings in simulation time [80]. The main problem with importance sampling is

to come up with an appropriate change of measure for the rare event simulation problem at hand [18].

For more information on this technique see [59] in which the authors consider rare events associated with multi-dimensional light-tailed random walks, rare events associated with certain events involving heavy tailed random walks and rare events associated with queues and queuing networks. They also review the literature on developments of adaptive importance sampling techniques to quickly estimate common performance measures associated with finite state Markov chains. Another example of the implementation of the importance sampling technique is used in [45] on a hypothetical problem involving gray wolf management in the Great Lakes of the United States. Formulations for determining conservation plans for sensitive wildlife species must account for economic costs of habitat protection and uncertainties about how wildlife populations will respond. As a result, their problem was to determine the cost-efficient level of habitat protection that satisfies a viability constraint for a sensitive wildlife population. The approach to solving the habitat protection problem involves a search algorithm that systematically evaluates the costs and population survival probabilities of alternative protection plans and they found that the importance sampling technique produced the best results [45].

2.4.2 Ordinal Optimisation

Ordinal optimisation concentrates on finding good, better or best designs rather than on estimating accurately the performance value of designs [50]. Focus is on 'order' rather than 'value estimates' and 'good enough' rather than 'optimal' choices — in other words, this method concerns itself primarily with the ordering of solutions [114]. The goal of ordinal optimisation is to find good enough solutions for a complex optimisation problem. A good enough subset is the subset consisting of the top best ordered solutions. It is noted that it is difficult to find the subset for a simulation based problem unless all the solutions in the solution space are calculated and compared. The ordinal optimisation method uses rough estimates from a crude model to rank the solutions. However, even with the use of a crude model, estimation of performance values for all solutions of a large solution space may not be computationally feasible, which is one of the downfalls of this method. To attempt to overcome this downfall, the ordinal optimisation method uses a representative set of samples that are randomly selected in order to represent the original sample space [113]. This method has recently been applied to many areas in power systems such as optimal power flow with discrete control and bidding strategies of power supplier in markets [113].

It is noted that the Ordinal Optimisation procedure has two major differences to the ranking and selection procedure. The first is that ranking and selection procedures deal with selection from a set of usually less than 100, while in Ordinal Optimisation, subset selection from a set of representative samples of size in the thousands or more are considered. Second, the notions of softened criterion and ordinal comparison are not represented in the ranking and selection procedures [65].

[114] states that simulations conducted by various authors for a wide variety of problems

have shown that the rankings of alternatives stabilize before the convergence of the performance estimates. He also goes on to say that experiments have also suggested that it is possible to determine whether a solution is good or bad very early in the simulation with high probability. The features have served as advantages of the ordinal optimisation method.

In ordinal optimisation one tries to find these good, better or best designs in the top α percentile of all designs. There has been criticism in the fact that a design in the top α percentile may still be significantly inferior to the true optimum. However, given that there may be a limited computational budget and time the top α percentile of the designs is often what is wanted [114].

The way in which the ordinal optimisation method is carried out is as follows. The performance of the alternatives (the whole feasible space or a subset depending on the size of the problem at hand) are evaluated. The frequencies of occurrence of the various performances' values are then plotted in a histogram from the best to the worst. This histogram is denoted as the performance distribution for the system. Then, for a given performance distribution, one can estimate various statistical measures of the distribution via sampling [49]. Due to the stochastic nature of the problem, the sampling is further complicated by the fact that there is now added noise. However, it is still true that the accuracy of any estimate about the distribution based on these noisy samples is only a function of the size of samples and the magnitude of the noise and is independent on the size of the underlying population [49].

Techniques based on gradients, feasible directions and stochastic approximations suffer certain disadvantages that ordinal optimisation does not. These are defined as follows [114]:

- The $(n+1)^{th}$ iterate depends only on the information collected near the n^{th} iterate thus ignoring the information collected at the earlier points of the performance surface
- The performance is sensitive to the initial choice of the design parameter however they do not provide guidelines for appropriate selection of these initial points
- Since they are driven by local information they do not provide a global view of the performance response surface

Ordinal optimisation is very general because the solution is applicable to a very large class of problems, practical because it fulfills the engineering requirements for a useful and satisficing solution without any extreme claims on the degree of optimality of the solution. It is also a complementary method in that it does not replace but rather supplements existing techniques [114]. Another advantage of ordinal optimisation is that it is much easier to estimate approximate relative order than a precise value [27]. Lastly, estimation and comparison can be done based on very short observations or very approximate models and therefore considerable simulation work can be reduced [50].

An example in which ordinal optimisation is used can be seen in [54] to schedule semiconductor water fabrication.

2.4.3 Multiple Comparison with the Best

The goal of multiple comparison with the best (MCB) is to quantify the differences between the performance of systems and determine the best system from a set of alternatives [105]. MCB specifies the use of certain pairwise comparisons to make inferences in the form of confidence intervals about relationships among all designs [30]. In MCB procedures, inferences about the relative performance of all alternatives tested is provided as well as information about the relationship among the designs. This inference is critical if the performance measure of interest is not the sole criterion for decision making [18]. An example of this, as given in [18], is the expected throughput of a manufacturing system may be the performance measure of interest but cost of maintaining the system is also important.

The main idea behind this method is to run a number of replications of the system with different alternatives and make conclusions on a performance measure by constructing confidence intervals. These simultaneous confidence intervals are formed on the parameters $\mu_i - \min_{j \neq i} \mu_j$ for $i = 1, 2, ..., k$ where i is a given alternative and μ is the value to be optimised. The $(k-1)$ simultaneous confidence intervals bound the difference between the expected performance of each system and the best of the other systems [105]. MCB procedures include subset selection by identifying the set of design alternatives whose MCB confidence interval includes zero [105].

The MCB procedure can be implemented in a single run of each alternative under consideration which is important as the restarting of simulation experiments can be expensive [18]. Due to the fact that the MCB approaches optimisation problems as a statistical inference problem it does not guarantee a decision like other approaches such as ranking and selection [44]. [18] provides a list of references which show adaptations of the MCB procedure for different simulation problems.

2.4.4 Ranking and Selection

Ranking and selection procedures are designed for distinguishing among a given set of alternatives for a particular model. It is a method used in a finite space and involves selecting the best of a few alternatives, that is $\Theta = \{\theta_1, \theta_2, \theta_m\}$ where m is relatively small and therefore it is possible to evaluate every solution and compare the performance. In that sense, these procedures are not simulation optimisation procedures per se since they lack the distinctive search feature (most simulation optimisation procedures try to search efficiently through the given set to find improving solutions because exhaustive search is impractical or impossible) [29, 27]. The ranking and selection procedures are frequently employed for practical problems and focus on selecting an optimal input parameter [44]. This method is mainly used for problems that contain only a small number of feasible systems (generally less than thirty) [112]. Ranking and selection is also able to treat the optimisation problem as a multi-criteria problem and is generally used when some knowledge of the relationship among the alternatives is available [18].

There are a few disadvantages to this method in that in order to be able to use it one needs to be able to specify an indifference zone and confidence level to be used [27]. Another disadvantage is the requirement of independence over competing designs which prevents the use

of most variance reduction techniques in discrete event simulations [5]. Lastly, there is the fact that only a certain number of alternatives can be compared. The more alternatives included the more timely and costly the experiment.

R&S procedures can be grouped into three main categories: indifference zone procedures, subset-selection procedures and other procedures. When the decision involves selecting the best system design, the technique of indifference-zone ranking is often used. When the decision involves selecting a subset of system designs that contain the best design the technique of subset selection can be used [18]. In both cases the decisions are guaranteed to be correct with a pre-specified probability [18]. These two approaches can be described as follows:

- Indifference zone - with a probability p the given scenario will be the 'indifference zone' away from the optimum
- Subset selection - entails obtaining a subset of random size that contains the best system with user-specified probability p*.

Subset selection approaches would seem to be most useful when the number of choices is quite large, the goal being to reduce the large number to a manageable random but smaller number. Indifference zone approaches could then be used to select a single choice that is within a pre-specified difference from the true optimum [5]. For a detailed outline of these two approaches see [105].

Some articles in which this simulation optimisation technique of ranking and selection has been used is in [116] where a framework for combining the statistical efficiency of simulation optimisation techniques with the effectiveness of parallel execution algorithms and also in [16] where statistical procedures are developed that return the best system encountered by the search with a pre-specified probability.

2.4.5 Combination of Multiple Comparison with the Best and Ranking and Selection Procedures

In the last few decades, research has been done in combining the methods of R&S and MCB. R&S and MCB methods are particularly well suited for computer simulation. In particular, these procedures are statistically valid in the context of simulation as the underlying assumptions of normality and independence of observations can easily be achieved through appropriately batched output data or sample averages of independent replications, and through adequate assignment of the pseudo-random number generator seeds respectively. Furthermore, common random numbers, a variance reduction technique widely used in simulation can be used to enhance the efficiency of some of the procedures [105].

There has been a recent effort to combine R&S procedures with MCB procedures. The idea of combining these two approaches is appealing because such an approach not only selects the best system with prespecified confidence (R&S), but it allows inferences to be drawn about the relationships between systems that may facilitate decision-making based on secondary criteria that are not reflected in the performance measure selected (MCB). Also, combined R&S-MCB

procedures allow for both R&S selection and MCB inference with little or no additional computational overhead.

[76] shows that most indifference-zone procedures can simultaneously compute MCB confidence intervals corresponding to the indifference zone. Therefore, both indifference-zone ranking and MCB inference can be derived from the same experiment with a pre-specified MCB interval with width being the indifference zone size. The authors go on to describe four R&S-MCB procedures that depend on having normally distributed data, but do not require known or equal variances. The four procedures are listed below:

1. Rinott's Procedure
2. Dudewicz and Dalal's Procedure
3. Clark and Yang's Procedure
4. Nelson-Matejcik's Procedure (NM procedure)

[76] reports results that suggest that the NM procedure is superior to the other three procedures as it in terms of the total number of observations required to obtain the desired confidence level. As a result, the other three procedure will not be discussed. For more detail on these procedures see [20], [23] and [86].

The NM procedure assumes that the unknown variance-covariance matrix of the output from simulation runs, $\sum$, exhibits a structure known as spherity. More specifically, the spherity structure takes the form

$$\sum = \begin{pmatrix} 2\psi_1 + \tau^2 & \psi_1 + \psi_2 & \cdots & \psi_1 + \psi_k \\ \psi_2 + \psi_1 & 2\psi_2 + \tau^2 & \cdots & \psi_2 + \psi_k \\ \vdots & \vdots & \ddots & \vdots \\ \psi_k + \psi_1 & \psi_k + \psi_2 & \cdots & 2\psi_k + \tau^2 \end{pmatrix}$$

where $\tau^2 > \sqrt{k\sum_{i=1}^{k}\psi_i^2} - \sum_{i=1}^{k}\psi_i$ such that $\sum$ is guaranteed to be positive definite [76]. Spherity implies that $Var[Y_{ij} - Y_{lj}] = 2\tau^2$ for all $i \neq l$. This means that the variances of all pairwise differences across systems are equal , even though the marginal variances and covariances may be unequal. It is noted that although procedure NM is valid when $\sum$ satisfies spherity, [76] shows it to be extremely robust to departures from spherity. This has been a great advantage as often testing for spherity can prove to be difficult or not possible and therefore showing that the process is robust to departures from spherity means that it is more applicable.

[40] describes the following steps to the NM procedure:

1. Specify the indifference zone, δ, the acceptance probability, α and the initial number of replications, n_0. Let $g = T^{1-\alpha}_{k-1,(k-1)(n_0-1),0.50}$ is the $(1-\alpha)$-quantile of the maximum of a multivariate t random variable with $k-1$ dimensions, $(k-1)(n_0-1)$ degrees of freedom and common correlation 0.50.

2. Take independent and identically distributed samples $Y_{i1}, Y_{12}, ..., Y_{in_0}$ from each of the k competing systems using common random numbers across systems. Where Y_{ij} if the j^{th} replication of the i^{th} design.

3. Compute the sample variance of the difference under the condition of spherity as

$$S^2 = \frac{2\sum_{i=1}^{k}\sum_{j=1}^{n_0}(Y_{ij} - \bar{Y}_{i\cdot} - \bar{Y}_{\cdot j} + \bar{Y}_{\cdot\cdot})^2}{(k-1)(n_0-1)}$$

where $\cdot$ represents averaging with respect to that subscript.

4. Compute the final required sample size (constant for all k alternatives) as

$$N = max[n_0, \lceil (gS/\delta)^2 \rceil]$$

5. Take $N - n_0$ additional independent and identically distributed observations from each system, using common random numbers across systems

6. Compute the overall sample means for each system as

$$\bar{Y}_{i\cdot} = \frac{1}{N}\sum_{j=1}^{N} Y_{ij} \text{ for } i = 1, 2, ..., k$$

7. Select the system with the smallest $\bar{Y}_{i\cdot}$ as the best alternative

8. Simultaneously, form the MCB confidence intervals as

$$\mu_i - \min_{j\neq i}\mu_j \in [-(\bar{Y}_{i\cdot} - \min_{j\neq i}\bar{Y}_{j\cdot} - \delta)^-, (\bar{Y}_{i\cdot} - \min_{j\neq i}\bar{Y}_{j\cdot} + \delta)^+] \text{ for } i = 1, 2, ..., k$$

where $-x^- = min[0, x]$ and $x^+ = max[0, x]$

The main advantages of the NM procedure is that it is easy to implement and using a combined R&S-MCB procedure the analyst gains inferences on systems other that the best, which may lead to the selection of an inferior system (if it is not inferior by much) based on some secondary criterion not reflected in the performance measure of interest [72].

2.5 Response Surface Methodology

Response surface methodology (RSM) is a collection of statistical and mathematical techniques which are used to explore unknown surfaces and optimising stochastic functions and is a method often implemented for optimisation of stochastic simulation models [41]. This is a method that is mainly aimed at models where the calculation of the corresponding stochastic objective function is very expensive or time consuming to implement [27]. Also, due to the fact that RSM is sequential in nature, with each successive experiment building on the results and insights of

earlier experiments, it is highly applicable to simulation due to the inherent nature of the ease with which simulation models can obtain data [53].

The main goal of RSM is to obtain an approximate functional relationship between the various input variables and the corresponding output objective function [27]. This simulation optimisation method is widely used in engineering and has also been seen to be a valuable technique in developing new processes and systems, optimising their performance and improving the design and formulation of new products [57].

RSM is a stepwise method and is based on the approximation of the stochastic objective function by a low order polynomial on a small sub region of the domain. The coefficients are estimated by ordinary least squares applied to a number of observations of the stochastic objective function. Based on the fitted polynomial, the local best point is derived which is used as a current estimator of the optimum and as the center point of a new region of interest where again the stochastic objective function is approximated by a low order polynomial [41]. This process usually starts off by using first order regression functions due to the fact that they are sufficiently robust to allow the estimation of the interaction effects as well as the linear effects [90]. First order polynomials are used until they no longer adequately estimate the response surface or after the have reached a point close to the optimum. At this stage, higher (second order) degree regression function are used [10]. One way to establish the lack-of-fit of the model is to use the coefficient of determination. There are other diagnostic measures available and are outlined in [62].

There are many advantages to RSM defined as follows:

- RSM requires a small number of simulation experiments relative to many gradient based methods [18].
- There are a lot of well-known and well-studied statistical tools such as regression analysis and analysis of variance contained within RSM.
- RSM allows for a lot of transparency for the user [27].
- RSM gives a fast approximation to the model, which can be used to identify important variables, visualize the relationship of the input to the output and quantify the trade-offs between multiple objectives [57].
- The polynomials that are fit can be fit to transformations of the variables [27].
- The automated RSM optimisation methods are less time consuming since there is no need to interfere in the optimisation process and therefore serves as a great advantage in large scale time consuming applications [41].

As with advantages, there are also disadvantages to RSM. A few significant ones are:

- The generality of the approach can turn into a drawback if not applied properly [27].

- Both local and global optimum locations can be generally identified. However, since a limited number of data points are used in order to generate the response surface, the surface approximates the actual behaviour and the results are similarly approximated though for many practical problems this is adequate enough [57].

- The performance of RSM strongly depends on the magnitude of the experimental error. RSM is more successfully applied when the experimental error is small. The stochastic nature of the simulation can cause volatility in the simulated responses which can cause large experimental error. This problem is aggravated when the experimental points under investigation are too far apart [90].

- RSM is mostly a continuous state space methodology and works best on problems with a small number of variables and on problems with special structures and therefore is not always the most appropriate tool for discrete-event simulation optimisation [64].

A compromise response surface method (CRSM), which is a modification of the response surface method, has been developed to reduce the simulation runs needed for generating the response surface and this increases the efficiency of the optimisation strategy. This approach has been used for an impingement CVD reactor for thin film fabrication [57]. Another modified type is generalised response surface methodology (GRSM) which is a novel general-purpose heuristic. GRSM combines the estimated gradients with mathematical programming findings to estimate a better search direction than the steepest ascent direction used by RSM. Both GRSM and RSM estimate local gradients to search for the optimum. These gradient estimates use locally fitted first-order polynomials. GRSM, however, combines these gradients with Mathematical Programming findings to estimate a better search direction than the steepest ascent direction used by RSM. Like Mathematical Programming, GRSM allows multiple responses, selecting one response as goal and the other responses as constrained variables. The (deterministic) input variables may also be subjected to constraints. Finally, GRSM uses the estimated gradients in a bootstrap procedure for testing whether the estimated solution is indeed optimal [62]. Though GRSM has been developed for random simulations, it can be easily adapted for deterministic simulations and real-world systems [62].

RSM provides a very general methodology for optimisation via simulation and therefore it should also be used with other techniques to improve efficiency [5]. [79] implements an efficient optimisation methodology using RSM and genetic algorithms to minimize warpage of thin shell plastic parts manufactured by injection molding.

2.6 Simulation Optimisation Software

Since simulation optimisation has become more well known, it has prompted the inclusion of optimisation algorithms into simulation packages. This means that nearly every commercial discrete-event or Monte Carlo simulation software package contains an optimization module that performs some sort of search for optimal values of input parameters rather than just

performs pure statistical estimation. Practically, this has really been possible due to the fact that computational power has increased.

Most of the optimisation engines that are built into simulation packages are based on evolutionary approaches. There is one exception, however, which is the optimisation algorithm in WITNESS which is based on search strategies from simulated annealing and tabu search. Table 2.2 shows the main simulation optimisation software that are available. The table also shows which optimisation strategies the packages are based on, as well as a list of simulation software for which they are built. A URL is is given for where each of these software packages can be accessed.

Optimiser	Technology	Simulation Software	URL
OptQuest	Scatter search (Tabu search)	AnyLogic, Arena, Crystal Ball, CSIM19, Enterprise Dynamics, Micro Saint, ProModel, Quest, SimFlex, SIMPROCESSM, SIMUL8, TERAS	www.opttek.com/OptQuest
Evolutionary Optimser	Genetic Algorithms	Extend	www.extendsim.com
Evolver	Genetic Algorithms	@RISK	www.palisade.com/evolver
AutoStat	Evolution Strategies	AutoMod	www.appliedmaterials.com
WITNESS	Simulated Annealing and Tabu Search	L-SIM	www.lanner.com/en/media/witness

Table 2.2: Commercial Implementations of Evolutionary Approaches to Simulation Optimization

It is noted that R is used in this thesis instead of commercial packages. The main reason for this is so that there is direct control of the way the simulation optimisation takes place and also so that the simulation optimisation can be completely tailored to the simulation model at hand.

2.7 Conclusion

In this literature review, various simulation optimisation techniques were discussed. The different simulation optimisation techniques were broken down into four main categories, heuristic solution methods, gradient based search methods, statistical sampling and response surface methodology. For each technique, advantages and disadvantage were given as well as a reference to where the technique was used in a simulation optimisation context. It is seen that there are a many techniques for simulation optimisation available and that each technique is tailored to a specific type of problem. [29] defines some key issues in simulation optimisation and are listed below. There is the constant challenge of keeping these issues in mind as well as trying

to find the optimisation technique best suited to the simulation model.

- The neighbourhood definition
- Mechanism for exploration/sampling especially how previously generated (sampled) solutions are incorporate
- Determining which candidate solution(s) to declare best
- The computational burden of each function estimate (obtained through simulation replications) relative to search (the optimisation algorithm)

Simulation optimisation continues to be an active field of research with new techniques, as well as modifications of current techniques, constantly emerging.

Now that the available techniques have been explored they will be implemented. The next chapter outlines a test case simulation model onto which certain optimisation techniques will be implemented (justification of the choice of techniques is also provided). By exploring these techniques, one will be able to understand their functionality better. These techniques are then applied to a more detailed real life simulation model. It is noted, however, that there is no guarantee that the algorithm will perform well, or badly, in both the test case and real life simulation model.

Chapter 3. Simulation Optimisation of Test Case Simulation Model

3.1 Outline and Aim

The aim of this chapter is to test different optimisation techniques on a simple stochastic simulation model. A stochastic simulation model is different to a deterministic one in the sense that ranges of values for each variable is used instead of single estimates to represent the value of each variable. A simple test case simulation model that contains inherent randomness is created for this purpose. Different optimisation techniques are applied to this test case simulation model. The results of each technique are then looked at to see how the technique handles the inherent randomness in the simulation model. Finally, the optimisation techniques are compared to see which technique, if there is one, produces the best results. The five optimisation techniques that are tested and compared are the genetic algorithm, simulated annealing, nested partitions, Nelson-Matejcik's procedure and the ordinal optimisation method.

3.2 Test Case Simulation Model

3.2.1 Description of Test Case Example

An important goal of the optimisation studies was to investigate the potential for application of heuristic optimisation methods to goal seeking in simulation models, for purposes of policy analysis. The ultimate aim is to apply such methods to large simulation models being operated within the energy sector.

Before any optimisation techniques can be implemented a simple test case simulation is created. The simulation was modelled on the functioning of Open Cycle Gas Turbine (OCGT) plants. OCGTs function in the electricity sector and their main purpose is to serve as electricity providers in an emergency situation in order to supplement the amount of electricity through a short lead-time. It is noted that this is a test case simulation and is very much a simplification of real life functioning of OCGT plants and therefore must not be seen as an accurate portrayal of OCGT plants. In this simulation model many factors that play a real life role have not been considered and have therefore been excluded from the model or agglomerated into one. As such this model cannot be used to make any real life predictions or decisions and purely serves for illustrative purposes.

The basic outline of the simulation model is as follows: it considers policy decisions on fuel orders based on forecasted demands some months into the future, after which actual demands are simulated to generate changes in stocks, fuel shortages and unnecessary burning of excess fuels. The policies considered by the model relate decisions made (fuel orders) to parameters of the probability distribution on forecasted demands. Throughout the running of the simulation model the costs that are incurred are measured.

Figure 3.1 shows how the simulation model performs when in an optimisation context. The simulation model is run with a defined set of model parameters. This produces a goal function value which then undergoes an optimisation process whereby the model parameters are changed. The process then repeats itself until a satisfactory solution is reached.

At this point, it is worth noting that specific terminology is used throughout the implementation of the simulation optimisation methods. When referring to a run of a simulation model this means taking the input values, running the simulation model and then generating the output values. One run of the simulation model is one cycle of what is shown in figure 3.1 whereas these many iterations (cycles) make up one run of the simulation model.

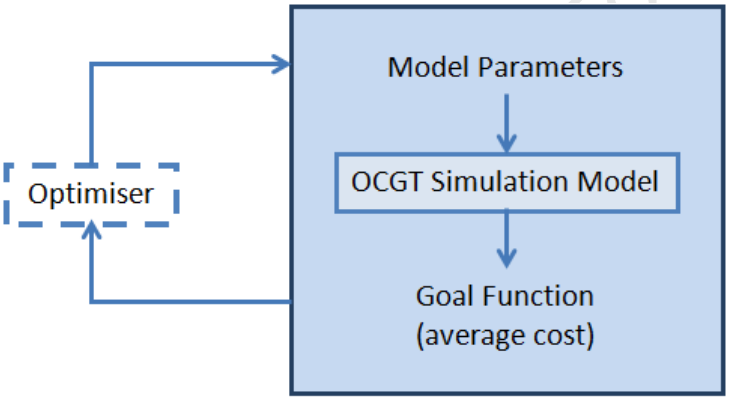

Figure 3.1: Cycle of simulation optimisation model when optimising test case simulation model

A description of this simplified OCGT simulation model is given below: The model iterates for n periods of time and works as follows:

- Demand in any one time period t is assumed to be log-normally distributed, i.e. the natural logarithm of demand is normal with mean μ_t (possibly varying with time t) and standard deviation σ_t (also possibly varying with time t). The resulting mean and variance of demand can then be shown to be respectively $m_t = exp(\mu_t + \frac{\sigma_t^2}{2})$ and $s_t^2 = exp(\sigma_t^2 - 1)exp(2\mu_t + \sigma_t^2)$. This structure is deemed to be realistic and rich enough to form a meaningful test case, while remaining simple to implement in numerical experiments.

- It is assumed that at the order time (at time $t - k$ say), the parameters of the lognormal distribution for time t are known but the actual demand is not known. The decision on quantity of fuel to be ordered for delivery at the start of period t needs to be a function of these parameters. For testing purposes, a simple function is adopted, for example of

the form $\alpha m_t + \beta s_t$, and the aim is to choose the policy parameters α and β in order to minimise expected costs. It is that the range over which the values of α and β will be looked at is $[0, 1]$ as it is unlikely that these values will be optimal outside of this range.

- In each time period, a random demand is generated from the lognormal distribution. At the end of the time period, the indicated closing stock of fuel would be the opening stock plus the order received less the actual demand. However:
  - If the nominal closing stock is negative (i.e. demand is not met), the stock is set to zero, and a cost is incurred (representing perhaps costs of power outages and/or of emergency fuel deliveries);
  - If the nominal closing stock is greater than maximum storage capacity, then the stock is set to this capacity, and a cost is incurred (representing perhaps costs of disposing of excess fuel and/or running of the turbines when not economical).
- The process is simulated for a given period of time over which all costs are accumulated. Minimization of these costs becomes the objective of the optimisation step for selection of the policy parameters α and β .

Figure 3.2 gives a graphical representation of the operation of the model.

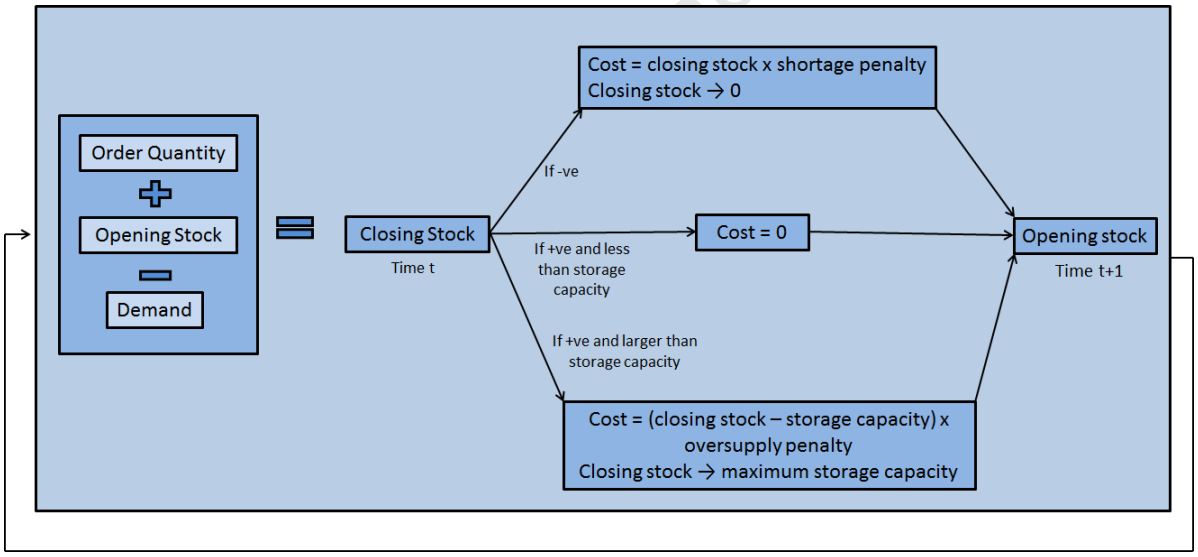

Figure 3.2: Open Cycle Gas Turbine Simulation Model

3.2.2 Simulation Model

The simulation model was built using R [84]. The computer that was used to run all the simulation optimisation models is a Intel(R) Core(TM) i5 CPU processor with and 8GB installed memory (RAM).

3.2.3 Variables

There are many parameters that could be used as control variables. For the purposes of this simulation model these parameters are defined as simply as possible. One assumes that the shortage penalty and the oversupply penalty that are incurred remain constant throughout the time the simulation model is run. For the purposes of this example, the shortage penalty was defined to be larger than the oversupply penalty due to the fact that it is assumed that a shortage will have more detrimental effects than an oversupply. Also, the maximum storage capacity is kept constant. One notes that everything is represented in terms of arbitrary 'units'. The values that were selected for these parameters are defined as follows:

- shortage penalty cost = 2 per unit under of fuel supplied;
- oversupply penalty cost = 1 per unit of fuel oversupplied;
- storage capacity = 0 (this is later changed to be 3 units, a value which was arbitrarily chosen, and is explored in more detail further on).

In order to ensure that the simulation model is realistic and rich enough to form a good case study the parameters of the lognormal distribution, which define the demand at a given time period, are allowed to fluctuate from one moment in time to the next. As previously mentioned, the natural logarithm of demand is normal with mean μ_t and the standard deviation σ_t both varying with time t. As a result, the mean and variance of demand can then be shown to be respectively $m_t = exp(\mu_t + \frac{\sigma_t^2}{2})$ and $s_t^2 = exp(\sigma_t^2 - 1)exp(2\mu_t + \sigma_t^2)$. At time period t of the simulation model the numeric values of the variables are set as follows:

- The value of μ_t is randomly generated from a normal distribution with mean 0 and standard deviation of 0.5.
- The value of σ_t is generated through the use of the coefficient of variation which, for a lognormal distribution, is defined as $exp(\sigma^2) - 1$. In this case study, the value of the coefficient of variation at time t, CV_t, is randomly generated from a uniform distribution with a minimum value of 0.5 and maximum value of 2. The value of σ_t is then defined as $\sqrt{log(CV_t^2 + 1)}$.
- From the values defined above m_t and s_t can be calculated such that $m_t = exp(\mu_t + \frac{\sigma_t^2}{2})$ and $s_t^2 = exp(\sigma_t^2 - 1)exp(2\mu_t + \sigma_t^2)$.

3.2.4 Order Policy

The order policy was defined to be a simple function of $\alpha m_t + \beta s_t$ where the parameter values α and β (the parameters that are to be optimised at a later stage) each assumed to be constructed to have a range of $[0, 1]$. This order policy was purely selected for its simplicity and in no way portrays a real life order policy.

3.2.5 Output Criteria

Only one variable is measured and recorded throughout the running of the simulation model. This is the cost that is generated at each time period (which results from when there is a shortage or an oversupply of fuel). Once the simulation model has been iterated the desired number of times an average cost value is calculated. This is defined as the cumulative cost generated from that run of the simulation model divided by the number of time periods the simulation model was iterated for. It is noted that this average cost value is what the optimisation model will ultimately try to minimise.

3.2.6 Simulation Horizon

When selecting the simulation horizon a trade-off has to be made between the accuracy of the results (the longer the simulation model is run for the more accurate the results) and the time the simulation model is run for (the shorter the simulation time the more desirable). In order to decide on a simulation horizon the simulation model was run for various numbers of periods (using the same parameter values) and the average cost was measured throughout the during of the simulation. It was seen that running the simulation for 1000 time periods allowed the average cost value to reach convergence as well as not taking too long for the model to run. The time it takes to run the simulation model with 1000 iterations is 0.5 seconds. Figure 3.3 shows the average cost for 9 runs of the simulation model. From this figure it can be seen that the average cost quantity has converged adequately by the 1000th iteration. Also, it can be seen that the model takes between 150 and 200 iterations to reach a rather reasonable steady state.

3.2.7 Plot of Output Function

As there are only two parameters that require optimisation, a plot of the output function can be constructed. This is helpful in understanding the possible output values for a given combination of parameter values. Since this is a simulation model, the output generated will be different every time the model is run and therefore so will the plot of the output function. As a result, an average output function was plotted based on the results of many simulation model runs. The following steps were taken to generate the plot:

1. Generate 50 α and 50 β values. For both of these parameters, their defined range is [0, 1]. This range is divided into 50 intervals in order to get the values of α and β.

2. Construct α and β pairs using the above values. A total of 2500 values are formed and shown in figure 3.4. From this it can be seen that a good representation of possible α and β value combinations have been used.

3. Run the simulation model 10 times for each of the α and β pairs and take the mean average cost that was generated for that pair over the 10 runs.

4. Construct a 3-dimensional graph showing the output value for each of the α and β pairs.

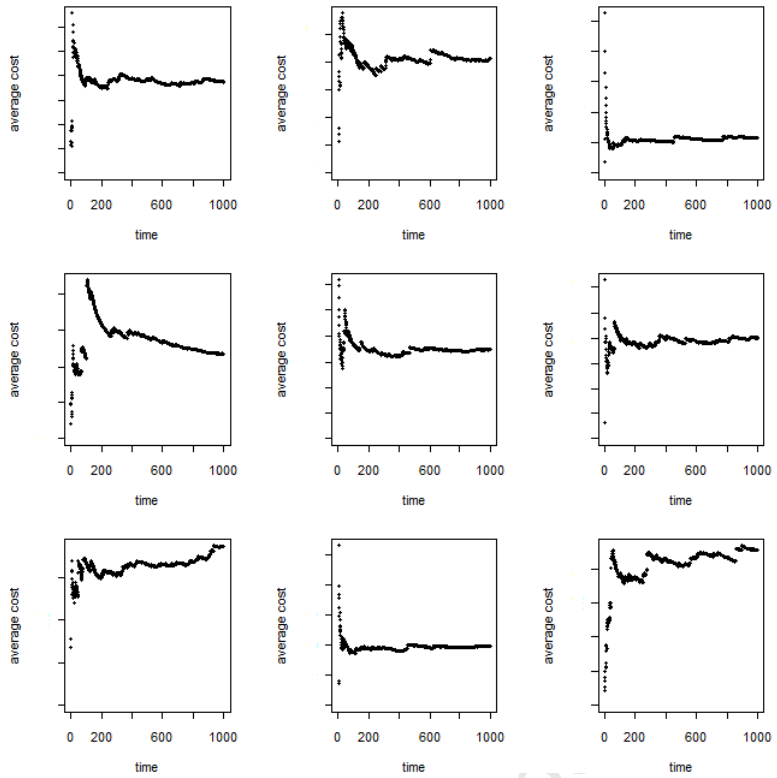

Figure 3.3: Average cost values over time for 9 runs of the simulation model

Figure 3.5 represents the mean average cost for various pairs of α and β values for 10 simulation runs. It can be seen that pairs of α and β values where both α and β take on small values as well as when they both take on large values generate a high average cost. It can also be seen that the minimum average cost is situated close to the equation $\beta = 1 - \alpha$ where α and β are defined over the range $[0, 1]$. If the optimisation methods work correctly, this relationship would be captured. Similarly, figure 3.6 shows the output generated for the different pairs of α and β values when the storage capacity is increase from 0 to 3 units.

3.3 Optimisation Problem

Now that the simulation model has been defined the optimisation techniques can be applied to the simulation model. The aim is to optimise the α and β parameters that are defined in the policy function in such a way that the best quantity of fuel can be ordered in order to satisfy the demand and limit the cost that results from an over or under supply.

3.3.1 Optimisation Techniques Used

The test case simulation model is defined to be a two-dimensional, continuous decision space model. With this in mind, certain simulation optimisation techniques are preferred to others. Five optimisation techniques were selected to be applied to the simulation model; three heuristic

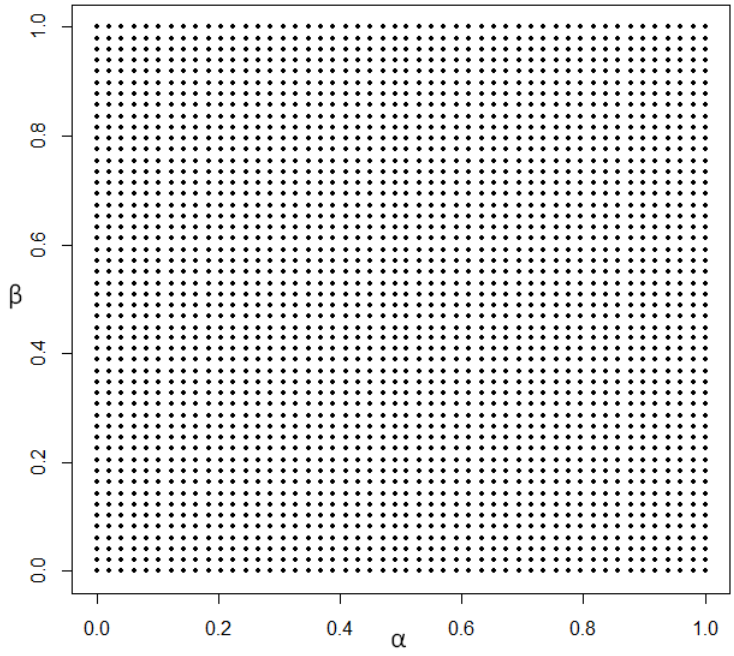

Figure 3.4: Selection of α and β values from feasible space

techniques and two statistical methods technique. The three heuristic techniques were genetic algorithm, simulated annealing and nested partitions. The statistical methods techniques are the Nelson-Matejcik's procedure, which is a combination of the multiple comparison with the best technique and the ranking and selection technique, and the Ordinal Optimisation method.

The genetic algorithm and simulated annealing techniques were selected as they have been shown in literature to produce good results and are well known methods, they are fairly easy to implement, and they are shown to be able to tolerate noisy, discontinuous and time varying functions [85, 8, 26, 34]. The nested partitions method was chosen as it is an interesting and fairly new method which proved interesting to explore and it has the advantage that it does not need a good starting point in order to be implemented [77, 86]. The Nelson-Matejcik's procedure was selected to be used as it combined two of the statistical methods into one and literature suggests that it is well suited to these types of problems [76]. Lastly, the Ordinal Optimisation method was chosen to be implemented as it is relatively easy to implement and focuses on order rather than value which can be seen as an advantage when dealing with simulation optimisation [50, 27].

The other optimisation techniques listed in the literature review were not selected for various reasons, justifications for these choices are given. The simplex search method was not selected due to the fact that this method is sensitive to the choice of initial parameter points [18]. A-teams were not examined directly as exploring all the optimisation methods available is a form of A-team by linking the human side with the computer side. Gradient based search methods were not applied as a whole but more specifically finite difference estimation was not implemented due to the fact that, although it is used in simulation optimisation, there

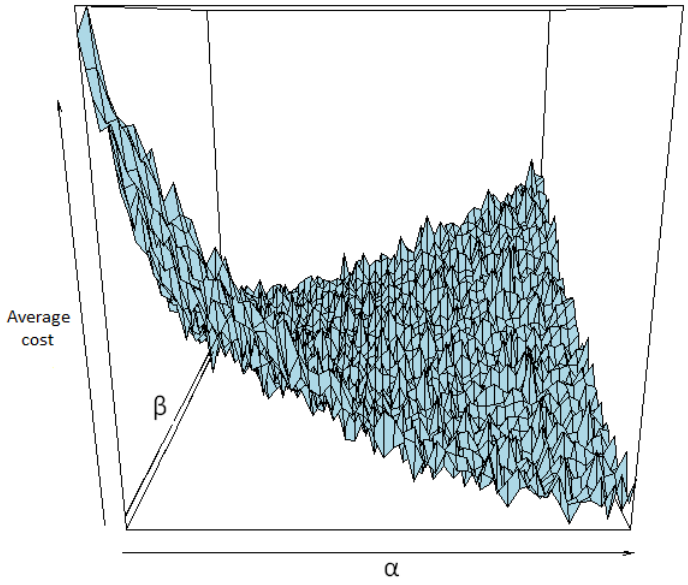

Figure 3.5: Plot of average cost for various combinations of α and β with no storage capacity included

are many disadvantages linked to this method [68, 5]. Infinitisemal perturbation analysis was not implemented. This was due to, firstly, this method struggles to handle 'bad' performance measures. Secondly, the simulation model that is being delt with runs fast enough that one does not need to estimate the values from a single run. Lastly, this method requires thorough knowledge of the simulation model which although holds true for the test case simulation model being dealt with will not hold true when looking at the real life simulation model [9]. Likelihood ratio estimators method was not used as having large variance associated with this method makes it undesirable [68]. Stochastic approximation was not implemented as this method was mainly designed for continuous variables are although research has started on handling models with discrete parameters it is still in the early stages [32]. Importance sampling is mainly used for simulation models involving rare events and therefore not implemented [18].

3.3.2 Genetic Algorithm

Preliminary Information

The Genetic Algorithm (GA) is a process that aims to imitate the process of natural evolution. The concept of GAs is outlined in the literature review while figure 3.7 gives a brief outline of how this process works.

Prior to implementing the genetic algorithm, various decisions need to be made with regard to the way in which the algorithm will run. These decisions relate to the following:

- Initial population creation.
- Population size.

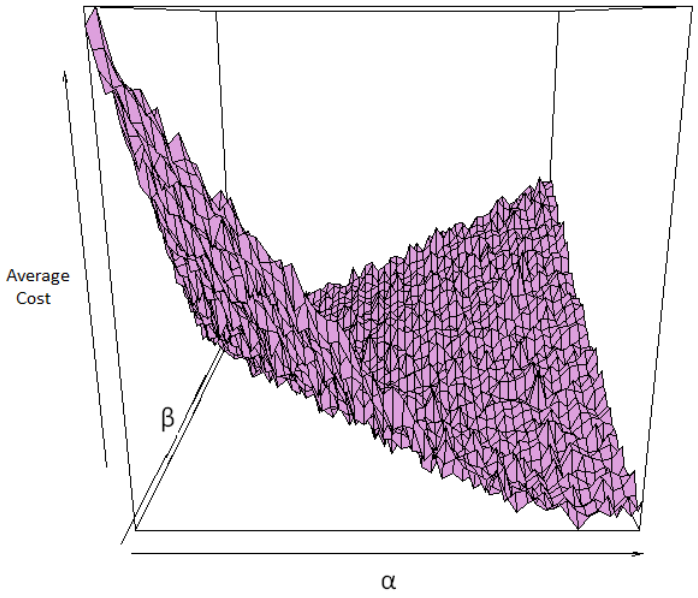

Figure 3.6: Plot of average cost for various combinations of α and β with storage capacity = 3 units

- The type of string coding that will be used for the parameters.
- Children creation and crossover mechanism.
- The mutation rate and the point at which the mutation takes place.
- Selection mechanism.
- Stopping criteria and optimal value selection.

These are discussed in more detail below.

Initial Population

The initial population is randomly generated which allows the entire neighbourhood of potential parameter values to be explored. The parameter values are allowed to take on any value within their range. This means that the initial α and β values can take on any value within the interval $[0, 1]$ over which they are defined. Also, the fact that the initial population is randomly selected means that the initial parameter values have the potential to be diversified and the whole feasible solution space to be explored.

Population Size

The population in each generation is made up of two aspects. The first is the number of parents and the second is the number of children. In this particular example, the number of parents, say x, is set equal to the number of children. The initial population size is equal to the parent size. From these parents, children are made, creating a population that is double the size of the initial population, $2x$. Therefore, the population size of each generation is of size $2x$. When creating the next generation, only x individuals from the current population progress to the

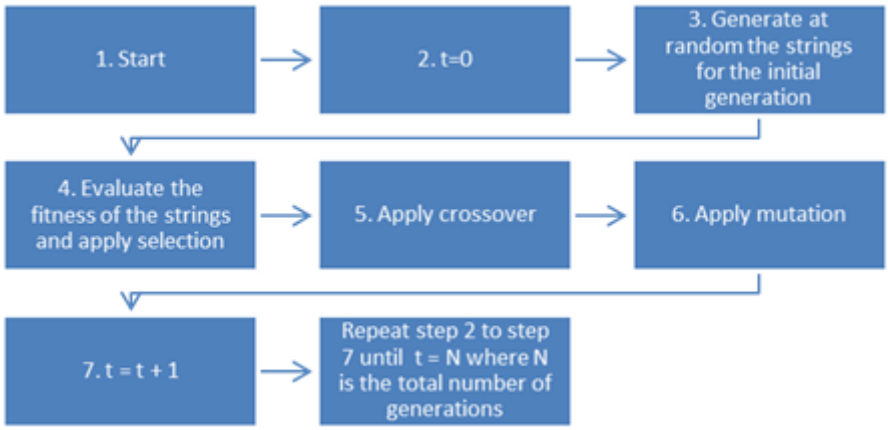

Figure 3.7: Outline of the Genetic Algorithm

next generation therefore creating the initial population for the following generation and the process of creating children is then repeated.

The type of string coding that will be used for the parameters

In order for the genetic algorithm to take place, more specifically the crossover and the mutations operations, the parameter values need to be encoded into a string form. There are many types of coding available; this section aims to explain the logic behind the choice of binary string encoding for the parameter values and then how the string code can be decoded to give the actual parameter values. As previously stated, there are two parameters being optimised; α and β, both having a range of $[0, 1]$. It is assumed that the accuracy of the α and β parameters up to 5 decimal places is sufficient.

In order to carry out the encoding over the parameters' range of [0, 1] with accuracy up to 5 decimal places, the range needs to be divided into 100 000 equal sized intervals. It is noted that $2^{16} < 100000 < 2^{17}$. This indicates that in order to reach the level of precision that is required each parameter needs to be coded using 17 bits i.e. each parameter has 17 positions that can take on a value of either 0 or 1. In order to decode a parameter the following formula is used

$$x_i = x_L + \frac{\sum_{i=0}^{bits-1} 2^i a_i}{2^{bits} - 1} \times (x_U - x_L)$$

Where x_i is the decoded parameter value, x_L is the lowest value that the parameter can take (0 in this case), x_U is the highest value the parameter can take (1 in this case), bits is the number of bits that the parameter has (17 in this case) and a_i is the bit value in the ith position (either 0 or 1) [47]. An example of a decoded string is given below:

$$00011100011100001 \longmapsto 0 + \frac{2^3 + 2^4 + 2^5 + 2^9 + 2^{10} + 2^{11} + 2^{16}}{2^{17} - 1} \times (1 - 0) = 0.52778$$

Children creation and crossover mechanism

In this particular genetic algorithm implementation x children are created in every generation.

These children are created using the following process which is repeated until the desired number of children have been created:

1. 4 parents are randomly selected
2. From these 4 individuals, the best performing two individuals are selected
3. These two individuals are then crossed over to generate 1 child

The crossover operation that takes place is defined as follows. The binary string representation for each of the parents is taken. If each parent has the same bit value at the same position in the string then the child gets that bit value at that position in its string. If the parents have different bit values at the same point in the string representation then a 0 or 1 is randomly generated and takes the corresponding position in the child's binary string representation. Figure 3.8 displays an example of the process. The binary values in blue correspond to the values that are the same in both parents. The values in black correspond to values that are different and as such the child value in that position takes on a randomly generated bit value.

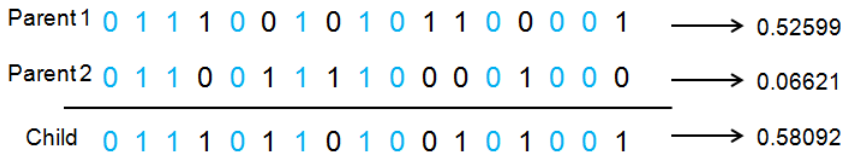

Figure 3.8: Example of a child being generated from two parents' string representation in the genetic algorithm

Mutation rate

The mutation of the string representation of the parameters is necessary when performing optimisation as it ensures that the algorithm does not converge to a local optimum result. In this particular case, the algorithm randomly generates $x \times mutation\ rate$ numbers from the range $[0, x]$ without replacement where x is the parent size of the population. These numbers correspond to the pairs of α and β parameters that are to get mutated. Once these individuals have been identified, 2 numbers are randomly generated from the range $[0, 17]$. These 2 numbers correspond to the parents in that particular generation whose position in the α and β string representation are to be mutated. If the binary value is 0 it changes to a 1 and similarly, if the value is a 1 it changes to a 0. Only one bit in the string of each parameter is ever changed. So for example, suppose that the initial population size is $x = 100$ and the mutation rate is 0.1. This means that 10 individuals will get mutated. These 10 individuals are randomly selected, for example 1, 4, 24, 34, 48, 49, 57, 87, 92 and 99. These individuals then have a bit in both their α and β parameters changed. Figure 3.9 on page 59 displays an example process for an arbitrary α parameter.

Selection mechanism

The role of a selection mechanism is to reject less fit individuals and select the fitter individuals to move onto the next generation (In this test case example, the lower the average cost value

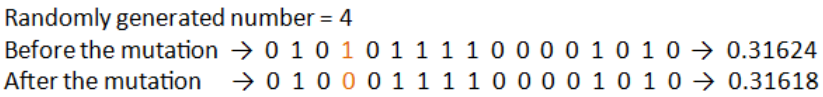

Figure 3.9: Example of a mutation in the genetic algorithm

the fitter the individual). After trying various selection mechanisms, the one chosen to be applied to this test case was the elitest selection mechanism. The way that this mechanism selects the individuals from the current generation to move onto the following generation is by comparing the fitness of all the individuals in the population, i.e. $2x$ individuals, and selecting the x individuals that performed the best. These x individuals then become the parents for the next generation. The advantage of this selection mechanism is that it ensures that the best performing individuals are guaranteed to move onto the next generation.

Stopping criteria and optimal value selection

There are many possible stopping techniques. For this particular simulation optimisation model the stopping technique used was that the algorithm would be terminated after the desired number of generations had been reached. When the simulation optimisation model has terminated, the optimal parameter values returned are those that generated the lowest average cost in the last generation of the model.

Simulation Optimisation

The genetic algorithm is now implemented on the simulation test case model. The following process takes place when the algorithm runs:

1. Randomly generate initial x parents' string representation.
2. Decode the string representation of the x parents to get the parameter values, α and β.
3. Run the simulation model for each of the x parents using their corresponding parameter values.
4. Generate x children from the parents in that generation through the crossover mechanism defined in the above section.
5. Apply the mutation mechanism described in the section above.
6. Decode the string representation of the x children to get their corresponding parameter values.
7. Run the simulation for each of the x children.
8. Select the best x individuals from the combination of parents and children in that particular generation using the selection mechanism described in the section above.
9. Repeat from step 4 until the stopping criteria has been satisfied.

In order to decide on the final model to run that generated the best results, various models with various specifications were tried. Firstly, different population and generation sizes were compared. Once an accurate population and generation size were selected the model was run with various mutation rates to see which rate produced the best results thereby giving the final model specifications. A model is defined as better than another if the mean average cost value for 10 runs of that model is smaller than another model's. The range of the parameter values for the 10 model runs are also considered when deciding the best model. It is noted that although running the models 10 times is not a lot, it is done to serve as an illustration.

Different population and generation sizes Table 3.1 below shows the various models that were tried with their various population and generation sizes. The mutation rate was set to 0.1 as a default value. This means that 0.1 of the children will undergo a mutation at each generation. This table also shows the length it takes to run one run of the simulation optimisation model.

Each of these models was run 10 times. The results are summarised and shown in Table 3.2. The table shows the two parameters, α and β as well as the average cost. The table contains the minimum and maximum value from the 10 runs of each of the 7 models as well as the range and the mean value achieved through these 10 runs of each model. From Table 3.2 the following can be seen from the results generated from 10 runs of each of the 7 models:

- From the results we can see that all the models produce fairly similar results in terms of the optimal average cost value as the mean average cost for all models are fairly similar. Comparing the best performing model (model GA7) with the worse performing model (model GA3) it is seen that they are statistically significantly different (p-value 0.002). However, comparing model GA7 and model GA2 it is seen that they are not statistically significantly different (p-value 0.71).

- The possible combination of α and β values that are optimal have a very large range for each of the models run which indicates that there are many possible combinations of parameter values which yield similar average cost values.

- The model that performs best in terms of average cost minimisation is model GA 7 followed by model GA 2. It is seen that although these models have the lowest average cost, the associated range from the 10 model runs is the largest for these two models. However, these models have some of the smallest ranges for the parameter values out of the 7 models that are tested.

- It is interesting to note that the model with the largest population size and number of generations (model GA6) does not perform the best.

- The final model that is chosen to be run is model GA 2. This is due to the fact that the results generated from this model are very similar to those generated by model GA 7 but the computational time for model GA 2 is less than for model GA 7. A small trade off is therefore made in favour of smaller computational time to accuracy of results.

Model	Population Size (parents+children)	Generation Size	Model run time (seconds)
GA 1	50 + 50	50	241
GA 2	100 + 100	100	932
GA 3	20 + 20	20	38
GA 4	200 + 200	50	1065
GA 5	20 + 20	100	182
GA 6	200 + 200	200	4096
GA 7	100 + 100	200	1935

Table 3.1: Model specifications for various population and generation sizes for a mutation rate of 0.1 and the elitist selection scheme for the genetic algorithm

Model	α				β				Average cost			
	min	max	range	avg.	min	max	range	avg.	min	max	range	avg.
GA 1	0.051	0.951	0.9	0.644	0.074	0.717	0.643	0.301	1.602	1.681	0.079	1.649
GA 2	0.535	0.975	0.44	0.698	0.064	0.416	0.352	0.22	1.448	1.646	0.198	1.588
GA 3	0.039	0.921	0.882	0.396	0.032	0.906	0.874	0.538	1.629	1.782	0.153	1.705
GA 4	0.419	0.923	0.504	0.708	0.081	0.526	0.445	0.235	1.542	1.658	0.116	1.613
GA 5	0.54	0.968	0.428	0.722	0.142	0.374	0.232	0.237	1.58	1.686	0.106	1.643
GA 6	0.228	0.923	0.695	0.679	0.005	0.584	0.579	0.236	1.542	1.639	0.097	1.591
GA 7	0.572	0.892	0.32	0.664	0.064	0.357	0.293	0.202	1.448	1.646	0.198	1.573

Table 3.2: Summary of results from 10 runs of each of the 7 models defined in Table 3.1 for the genetic algorithm

Model	Mutation rate	Model run time
GA 2.1	0	1000
GA 2.2	0.05	990
GA 2.3	0.1	976
GA 2.4	0.15	976
GA 2.5	0.2	1006
GA 2.6	0.3	1010

Table 3.3: Model specifications for various mutation rates with a population size made up of 100 parents and 100 children, a generation size of 100 and the elitist selection scheme for the genetic algorithm

Different mutation rates Now that the population size and number of generations have been decided upon (model GA 2) different mutation rates are tested and the results are compared. Table 3.3 outlines the model specifications with the different mutation rates.

Again, each of these models were run 10 times and their results were compared and are shown in table 3.4. From the results in table 3.4 it can be seen that very similar results are generated from the different mutation rates but that having a mutation rate of either 0.1 or of 0 produces the best results. One does not want a mutation rate of 0 because there will be no way to avoid getting stuck at a local optimum. This means that the mutation rate of 0.1 is chosen. A mutation rate of 0.1 means that 10 of the 100 children generated at each generation undergo a mutation. This means that a randomly selected bit value in both the α and β parameter of the child get changed to the opposite value.

Final model The specifications for the model used are defined as follows:

- Number of parents in each generation: 100.
- Number of children in each generation: 100.
- Number of generations: 100.
- Mutation rate: 0.1.
- Selection criteria: best 100 from the parents and children of each generation move onto the next generation.

Table 3.5 shows the output generated from 10 runs of the simulation optimisation model defined above. All the runs achieved convergence. From this one can see that the model converges to an average cost around 1.59. It is also seen that there does not appear to be an optimal solution for the values of the α and β parameters. From the results it seems as though there are many possible combination of α and β over their possible ranges that yield a very similar average cost value.

Model	α				β				Average cost			
	min	max	range	avg.	min	max	range	avg.	min	max	range	avg.
GA 2.1	0.196	0.889	0.693	0.702	0.035	0.625	0.59	0.212	1.558	1.661	0.103	1.588
GA 2.2	0.22	0.809	0.589	0.608	0.09	0.498	0.408	0.255	1.575	1.664	0.089	1.616
GA 2.3	0.535	0.975	0.44	0.698	0.064	0.416	0.352	0.220	1.448	1.646	0.198	1.588
GA 2.4	0.35	0.88	0.53	0.599	0.089	0.559	0.47	0.282	1.536	1.635	0.099	1.601
GA 2.5	0.169	0.876	0.707	0.605	0.046	0.777	0.731	0.288	1.546	1.638	0.092	1.594
GA 2.6	0.55	0.992	0.442	0.763	0.027	0.437	0.41	0.192	1.589	1.682	0.093	1.627

Table 3.4: Summary of results from 10 runs of each of the 7 models defined in table 3.3 for the genetic algorithm

Run	α	β	Average cost
1	0.602	0.151	1.601
2	0.927	0.187	1.610
3	0.975	0.114	1.602
4	0.535	0.416	1.625
5	0.585	0.270	1.448
6	0.582	0.357	1.606
7	0.688	0.064	1.571
8	0.728	0.300	1.646
9	0.675	0.266	1.633
10	0.684	0.077	1.538
Mean absolute deviation			0.200

Table 3.5: Output from 10 runs of the final genetic algorithm model

Run	α	β	Average cost
1	0.719	0.197	0.717
2	0.724	0.130	0.746
3	0.690	0.202	0.765
4	0.699	0.204	0.765
5	0.862	0.026	0.795
6	0.559	0.359	0.746
7	0.816	0.017	0.790
8	0.263	0.481	0.732
9	0.617	0.319	0.770
10	0.494	0.267	0.758
Mean absolute deviation			0.094

Table 3.6: Output from 10 runs of the final genetic algorithm model with storage capacity = 3 units

Changing the storage capacity The same simulation optimisation was now applied to the test case simulation model but this time the storage capacity was increased from 0 to 3 units. Table 3.6 shows the results from 10 runs of this model. From this one can see that the model converges to an average cost that has a average of 0.758 units over 10 runs. It makes sense that the average cost is lower if a storage capacity is included in the model as there is less of a chance of a shortage of fuel supply as there is potential fuel held in storage. At the same time, there is less of a chance of having over supply cost as the over supply of fuel can be held in storage. From the results one can see again that there are many possible combination of α and β values that yield very similar average cost values.

Conclusion

In conclusion, it is seen that the genetic algorithm achieves convergence when applied to the test case simulation example. After trying genetic algorithm models with different specifications it is seen that the model that produces the best results is model GA 2.2 which has number of parents in a generation = 100, number of children in a generation = 100, number of generations = 100, mutation rate = 0.1 and the best selection scheme is used. The average time that this model takes to run is 976 seconds. It is noted that possible results could be improved upon if the model was run with more generations but the slight improvement in results was not enough to compensate for the increase in computational time. Lastly, it is seen that there are many possible values that the α and β parameters can take on while still resulting in a converging average cost value. It does, however, appear as though the minimum average cost values are achieved when α is set to be larger than 0.5 and β is set to be a value smaller than 0.5.

3.3.3 Simulated Annealing

Preliminary Information

Simulated annealing is another commonly used metaheuristic designed to permit escaping from local optima by allowing worsening moves which facilitates in eventually finding the neighbourhood of the true optimum. Figure 3.10 gives a brief outline of how this process works. Again, this optimisation technique is aimed at optimising the α and β parameters in the test case simulation model, which are defined in the order policy, in order to minimise the average cost incurred.

Before the simulated annealing technique is implemented, the following model specifications need to be decided upon

- The neighbourhood structure
- The acceptance probability
- The initial temperature
- The cooling schedule

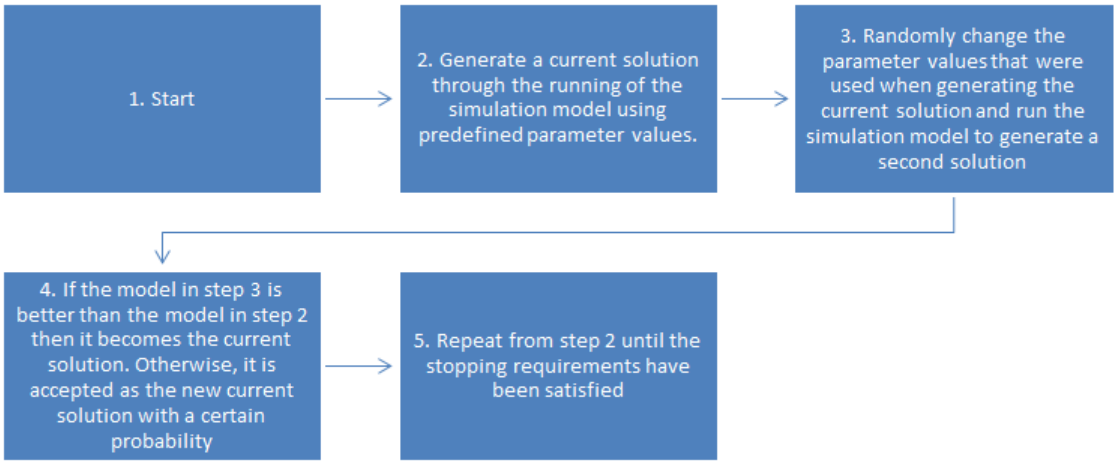

Figure 3.10: Outline of Simulated Annealing

- The stopping rule

The Neighbourhood Structure

The parameters used to calculate the second solution (step three in Figure 3.10) are generated from a region within the neighbourhood of the parameters used to generate the current solution (step two in Figure 3.10). The smaller the neighbourhood the faster the convergence as the search is more focused on the optimum area but it might mean that the convergence is to a local optimum. In this model the neighbourhood structure is always selected to be symmetric (it is seen that it allows calculations in the neighbourhood to be carried out quickly and efficiently [26]) and never allows parameters to be defined outside of their allowable range of $[0, 1]$. The neighbourhood structure is currently set to be ± 0.05 from the current parameter value. Different neighbourhood structures will be tested further on.

The Acceptance Probability

The acceptance probability is important as it is used to decide whether or not the second solution takes the place of the current solution even if it performs worse than the current solution (in the test case simulation example this would mean that the average cost for the second solution is higher than the average cost for the current solution). The main aim of the acceptance probability is to prevent getting stuck at local optima in the sense that it allows the algorithm to carry out worsening moves in the hope of finding the global optimum. The acceptance probability function is linked to the temperature of the system in such a way that as the temperature decreases (which takes place from iteration to iteration) so does the acceptance probability. This means that as the algorithm progresses the probability of accepting a worsening move decreases. The most commonly used acceptance probability function is the Boltzmann probability function which is defined as:

$$P(f(s), f(s^{'}), T) = \begin{cases} 1 & \text{if } f(s^{'}) - f(s) < 0 \\ exp(-\frac{f(s^{'}) - f(s)}{T \times K_B}) & \text{if } f(s^{'}) - f(s) \geq 0 \end{cases}$$

Where $f(s)$ is the current solution, $f(s^{'})$ is the second solution, T is the temperature and K_B is the Boltzmann constant (defined as 1.381) [1]. This probability is most often used as it is the one which most closely resembles the physical process of cooling molten metals on which simulated annealing is based. This probability function is the one used when applying the simulated annealing algorithm to the test case simulation model.

Temperature Selection

As the simulated annealing algorithm progresses, the temperature of the system is slowly decreased from iteration to iteration in order to force the average cost function to arrive at an optimum value.

In most of the literature it is seen that the initial temperature of the system needs to be high enough to allow virtually all transitions to be accepted at the beginning of the algorithm even if they have, for the purposes of this test case simulation model, a higher associated average cost value [7, 111, 24]. As previously stated, the Boltzmann acceptance criteria is being used as the acceptance probability. As a result, the Boltzmann distribution acceptance criterion should be approximately equal to 1 at the starting temperature. This allows for maximum exploration of the neighbourhood.

Therefore, using the definition of the Boltzmann probability above, in order to calculate the temperature at which the probability of accepting a solution is 1 when $f(s^{'}) - f(s) \geq 0$ we need to calculate $f(s^{'}) - f(s)$.

If running time was not a constraint, the simulation optimisation model would be run many times with different parameter values so as to get an average of the difference $f(s^{'}) - f(s)$. Due to the fact that running time is limited an approach was taken which, although saves on running time, does mean that the initial temperature value may not be very accurate. The conflict between cost versus accuracy is prominent in this area.

For each of the parameters α and β, their value can change by a maximum of 0.05 from one iteration to the next (due to the way in which the neighbourhood structure is defined). As a result the simulation model is run once with the parameter values of α and β both being 0.5 and then the simulation model is run again with the α and β values both being 0.55. The reasoning behind choosing value of 0.5 is that the range of possible values for α and β is $[0, 1]$ and therefore, if one assumes, a priori, that any value within that range can be correct with an equal probability it means that the mean parameter value is 0.5. The mean average costs for 100 runs of the test case simulation model are recorded for both the models; the one model having parameter values set to 0.5 units and the other model having parameter values set to 0.55 units, and the difference between the two values are then used to calculate $f(s^{'}) - f(s)$. The results are shown in table 3.7. It is noted that the same random number sequence was used when running both the two models shown in the table.

Now that these values of $f(s^{'}) - f(s)$ are known the initial temperature can be found. Figure 3.11 shows how the Boltzmann probability changes as the temperature value changes. From this graph it is seen that having an initial temperature above 30 gives an acceptance probability of close to 1. To proceed on the side of caution the initial temperature is therefore set to be equal to 40.

Model	α	β	Mean average cost
SA 0.1	0.5	0.5	2.098
SA 0.2	0.55	0.55	2.159
Difference	0.05	0.05	0.061

Table 3.7: Average cost values generated for different parameter values for simulated annealing

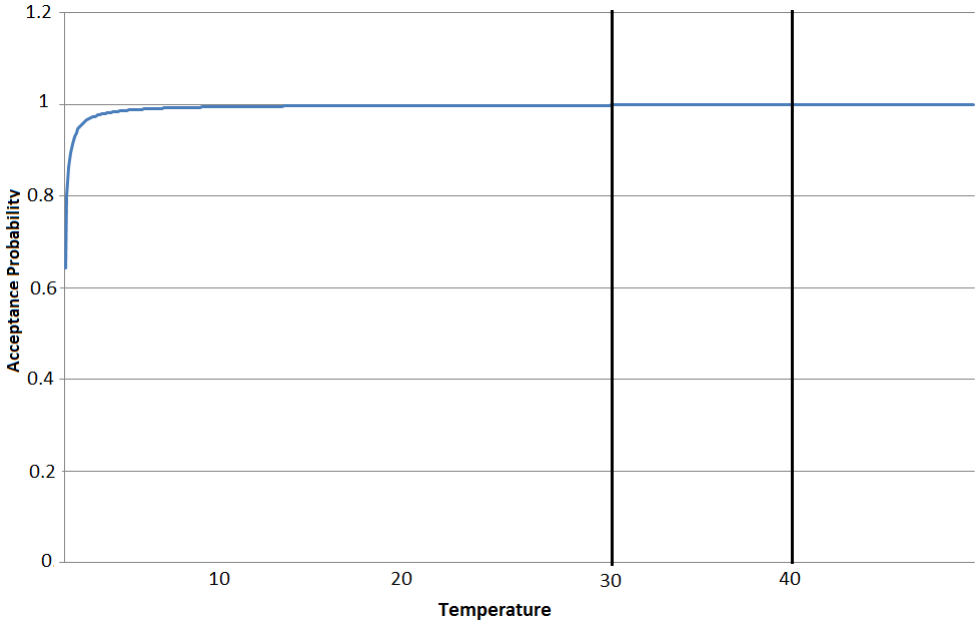

Figure 3.11: Graph showing the change in acceptance probability as the temperature changes for simulated annealing

A similar approach was taken when using a neighbourhood structure of 0.1 and 0.15. It was seen for these two neighbourhood structures that using an initial temperature of 40 was a good selection for their initial temperature as the acceptance probability was very close to 1 with this initial temperature.

Cooling Schedule

There are many possible cooling schedules available. Cooling schedules are defined to be non-increasing functions of time and are designed in a way such as to exclude almost all bad moves towards the end of the simulation optimisation algorithm. There is, however, always a compromise between the quality of the solutions obtained and the speed of the cooling schedule. The most common cooling schedule used is the geometric temperature reduction which is defined as $T_i = \gamma T_{(i-1)}$ where T_i is the temperature at the i^{th} iteration and γ is the cooling parameter [109]. Previous results from the literature have shown that setting the cooling ratio to lie between 0.8 and 0.99 yields the best results [110]. The higher the value of the cooling ratio, the slower the temperature will decrease to zero and as a result the model will need to be iterated for a longer period of time in order for the temperature to converge to 0. Figure 3.12 shows how the temperature changes at each iteration when a different cooling ratio is used. From

the figure it can be seen that the temperature converges to zero at a faster rate the smaller the cooling ratio. It can also be seen that if the cooling ratio is set to be smaller than 0.95 by the time the model has performed 150 iterations hardly any worsening moves will be accepted due to the fact that the temperature is so close to zero. One notes that the algorithm would have to be run for a very long period of time in order for the temperature to reach an actual value of zero. However, a temperature value close to 0 is seen to be sufficient. The two cooling parameters that are to be tested are 0.95 and 0.99.

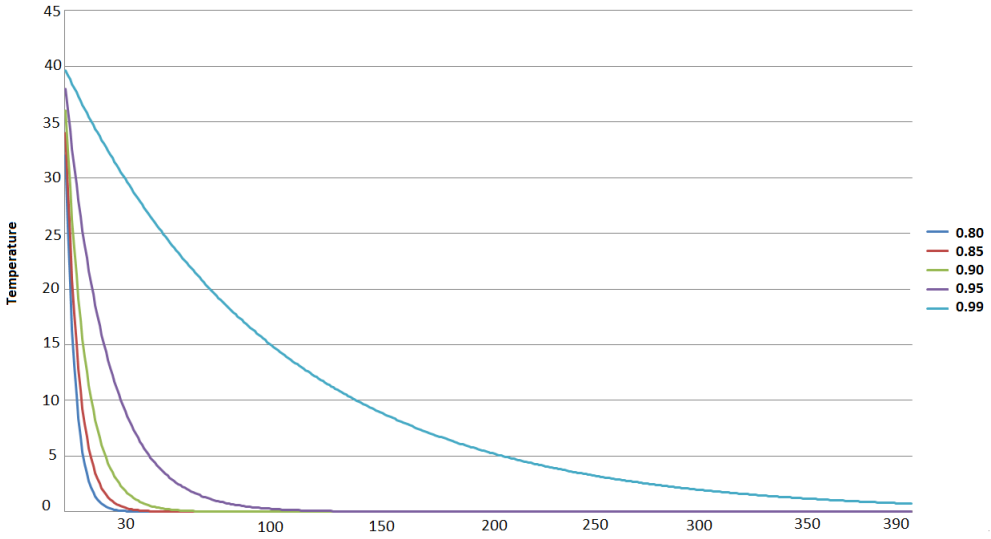

Figure 3.12: Change in temperature from one iteration of the simulation optimisation model to the next with varying cooling schedule parameters for simulated annealing

Stopping Rule

There are many types of stopping rules available. For example, the model can be terminated once the solution reaches convergence within a specific range of values. Another example, and the one that will be used when implementing the simulated annealing algorithm, is that the algorithm is stopped after a certain number of iterations have been carried out. For this simulation optimisation model, the simulated annealing algorithm is iterated 1000 times. An example of the output generated from 1000 iterations is shown in Figure 3.13. From this it can be seen that the parameters have almost converged after 300 iterations, although this is a local optimum as it can be seen that the parameter values converge to a different value which produces a lower average cost value at around iteration 500. This shows that having a large number of iterations is important. It is noted that by the time the simulation optimisation model has been iterated 1000 times that the temperature values has converged to 0 and therefore no worsening moves are accepted.

Simulation Optimisation

The simulated annealing algorithm is now explained in more detail.

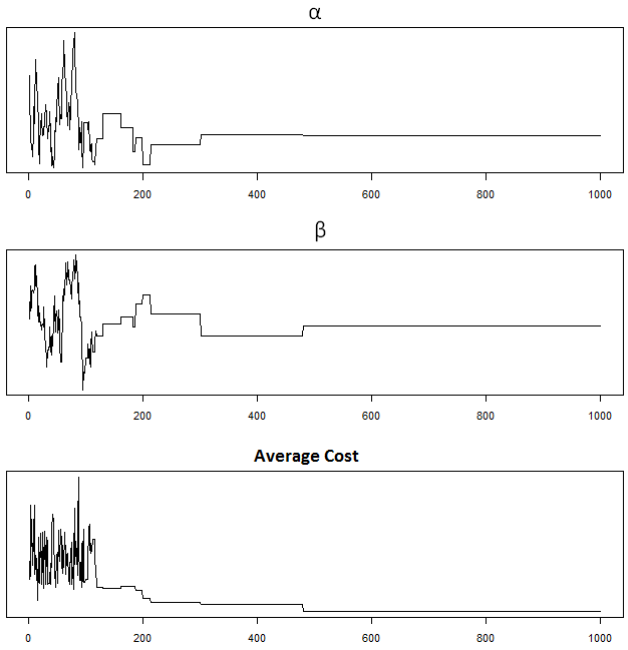

Figure 3.13: Graph showing an example of an implementation of the simulated annealing algorithm on the test case simulation model for simulated annealing

1. The initial model specifications are set
2. The number of times the simulation model is iterated (1000 times for the test case) and the number of times the optimisation aspect of the model is iterated (1000 times in this model) are then defined.
3. The initial temperature value is set
4. The parameter values (α and β) are randomly generated from the uniform distribution $[0, 1]$ since each parameter has an assumed range $[0, 1]$. Call these parameters α_0 and β_0.
5. The simulation model is then run with these parameter values and the final average cost (call this $(averagecost)_0$) as well as the parameter values are saved
6. The simulation model is then run again with different parameter values. Each of the parameter values are randomly generated from a uniform distribution that ranges from [max(0, parameter value from previous simulation run - ψ), min(1, parameter value from previous simulation run + ψ)] where ψ corresponds to the value associated in defining the neighbourhood structure. Call these newly generated parameters α_1 and β_1. The simulation model is run with these parameter values and the average cost value (call this $(averagecost)_1$) as well as the parameter values, are then saved.
7. The simulated annealing algorithm then iterates using the two average costs that were generated above. There are three possibilities that can take place

(a) If $(averagecost)_1 \leq (averagecost)_0$ then the parameters are redefined such that $\alpha_0 = \alpha_1$ and $\beta_0 = \beta_1$ and the current average cost is set to be $(averagecost)_1$

(b) If $(averagecost)_1 > (averagecost)_0$ then a random number, u, is generated and there are two possibilities that can happen

i. If $u < exp(\frac{(averagecost)_0 - (averagecost)_1}{temperature})$. Where $temperature$ is defined according to the relevant cooling schedule. Then the parameters are redefined such that $\alpha_0 = \alpha_1$ and $\beta_0 = \beta_1$ and we have that the current average cost is set to be $(averagecost)_1$

ii. Otherwise, the parameters are not reset

8. The process is then repeated from step 6 above until the stopping criteria is met.

9. At this point, the final parameter values should have converged to a constant value producing the minimum average cost.

In order to decide on the model to run that generated the best results, various models with different specifications were tried. Firstly, models with different neighbourhood structures were tried. Once the neighbourhood structure was attained the model was run with varying cooling parameter values to see which model to select. Again, a model is defined to be better than another if the mean average cost value for the 10 runs of the model is better. The range of the both the parameter values as well as the average cost value over the 10 model runs are also taken into consideration. It is noted that although running the models 10 times is not a lot, it is done to serve as an illustration.

Different neighbourhood structures Table 3.8 shows the various models that were tried with varying neighbourhood structures. In this case the default cooling parameter value was set to 0.95. The parameter value is set to be large due to the fact that this is a simulation optimisation and therefore there is inherent randomness in the average cost value. By having the cooling parameter quite large is prohibits converging to a local optimum too soon. As previously stated, having an initial temperature of 40 was suitable for the three neighbourhood structures that are to be tried. The geometric cooling schedule was set as the default cooling schedule as this is the one indicated in the literature to perform best.

Model	Neighbourhood structure	Model run time (seconds)
SA 1	± 0.05	97
SA 2	± 0.1	95
SA 3	± 0.15	83

Table 3.8: Models to be tested with different neighbourhood structures using the simulated annealing algorithm with the Boltzmann acceptance probability, an initial temperature of 40, a geometric cooling schedule and 1000 iterations

Each of these models were run 10 times and their results were compared as shown in table 3.9. This table shows the minimum and maximum values that both the parameters and the average

cost value take on over the 10 times each of the models were run. It also shows the range and the mean values that the parameters and average cost value had over the 10 runs of each of the models. From table 3.9 it can be seen that the model that produces the best results is

Model	α				β				Average cost			
	min	max	range	avg.	min	max	range	avg.	min	max	range	avg.
SA 1	0.061	0.813	0.752	0.421	0.069	0.990	0.921	0.525	1.635	1.895	0.260	1.726
SA 2	0.032	0.701	0.668	0.433	0.130	0.981	0.851	0.420	1.546	1.761	0.216	1.672
SA 3	0.049	0.736	0.687	0.441	0.177	0.972	0.795	0.479	1.599	1.758	0.159	1.683

Table 3.9: Summary of results from 10 runs of each of the 3 models defined in table 3.8 for simulated annealing

the model that has a neighbourhood structure of ± 0.1 (model SA 2) as it produces the lowest average cost and has the lowest range for the α and β parameters. A neighbourhood structure of 0.1 is most probably best suited as it allows the model not to get stuck at local optimum values. It is seen that there are many possible optimum α and β values which yield an optimal average cost result. It appears as though having a neighbourhood structure of ± 0.15 is too large and as a result means that the area around the true optimum is too greatly explored and therefore means that the optimal value is not reached in the defined running length.

Different cooling parameter values Now that a model SA 2 has been chosen, two different cooling parameters are tested to see which one produces the best results. Although it has been seen that iterating the simulated annealing algorithm 1000 times produces convergence the model was run with 2000 iterations of the simulated annealing algorithm and the different cooling parameter values. This was to see if the increase in computational time produced any significant benefit in the results achieved. These models are shown in table 3.10.

Model	Cooling parameter	Number of iterations	Model run time (seconds)
SA 2.1	0.95	1000	95
SA 2.2	0.99	1000	92
SA 2.3	0.95	2000	185
SA 2.4	0.99	2000	188

Table 3.10: Models to be tested with different cooling parameters using the simulated annealing algorithm with a neighbourhood structure of ± 0.1, a Boltzmann acceptance probability, an initial temperature of 40 and a geometric cooling schedule

These four models were each run 10 times and the results, shown in table 3.11, were compared. From these results it appears that having a cooling parameter of 0.95 is best suited to this model. It is also clear that increasing the number of iterations of the model to 2000 times is beneficial to better and more consistent results as the average and range of the average cost both decrease. However, it is noted that the results from model SA2.1 and SA 2.3 are not statistically significantly different (p-value = 0.01). Although this increase in number of iterations increases the run time, the run time is still quite small and therefore running 2000 iterations

Model	α				β				Average cost			
	min	max	range	avg.	min	max	range	avg.	min	max	range	avg.
SA 2.1	0.032	0.701	0.668	0.433	0.130	0.981	0.851	0.420	1.546	1.761	0.216	1.672
SA 2.2	0.074	0.944	0.871	0.464	0.058	0.826	0.767	0.508	1.588	1.795	0.208	1.699
SA 2.3	0.006	0.858	0.853	0.452	0.117	0.931	0.814	0.398	1.546	1.684	0.138	1.626
SA 2.4	0.175	0.923	0.748	0.487	0.051	0.774	0.723	0.498	1.588	1.723	0.135	1.662

Table 3.11: Summary of results from 10 runs of each of the 2 models defined in table 3.10 for simulated annealing

of the simulated annealing algorithm is not too damaging. There, from this the specifications associated with model SA 2.3 are chosen to create the final model.

Final Model The specifications for the model used are defined as follows:

- Number of iterations: 2000
- Neighbourhood structure: ± 0.1
- Acceptance probability function: Boltzmann probability function
- Initial temperature: 40
- Cooling schedule: Geometric
- Cooling parameter: 0.95

Table 3.12 shows the output generated from 10 runs of the simulation optimisation model defined above. For all 10 runs of the model, convergence of the parameter values and the average cost value were achieved. From this one can see that the model converges to an average cost around 1.63 units over 10 runs. It is also seen that there does not appear to be an optimal solution for the values of the α and β parameters as the range of the final parameter values is very large. From the results it seems as though there are many possible combinations of α and β values over their possible range that yield a very similar average cost value.

Run	α	β	Average cost
1	0.383	0.456	1.549
2	0.563	0.235	1.644
3	0.288	0.595	1.616
4	0.858	0.117	1.600
5	0.607	0.280	1.546
6	0.543	0.258	1.659
7	0.006	0.931	1.682
8	0.447	0.427	1.639
9	0.621	0.156	1.646
10	0.204	0.526	1.684
Mean absolute deviation			0.202

Table 3.12: Output from 10 runs of the final simulated annealing model

Changing the storage capacity The same simulation optimisation was now applied to the test case simulation model but this time the storage capacity was increased from 0 to 3 units. Table 3.13 shows the results from 10 runs of this model. From this one can see that the model converges to an average cost that has a average of 0.781 over the 10 runs of the model. It is seen that having a storage capacity decreases the optimum average cost value compared to when there is no storage capacity. Again, this makes sense as if there is stock in storage there is less of a chance of a shortage occurring while at the same time if there is an oversupply of stock, some can be kept in storage and as a result there will be less of an over supply penalty incurred. Although the average cost value seems to reach a similar value over the 10 model runs, the possible range of the parameter values is very large. There seem to be many combinations of the parameter values which yield very similar average cost values.

Run	α	β	Average cost
1	0.300	0.632	0.750
2	0.502	0.346	0.753
3	0.163	0.635	0.801
4	0.682	0.092	0.752
5	0.213	0.592	0.789
6	0.078	0.855	0.819
7	0.468	0.658	0.883
8	0.515	0.348	0.718
9	0.415	0.541	0.736
10	0.653	0.336	0.805
Mean absolute deviation			0.117

Table 3.13: Output from 10 runs of the final simulated annealing model with storage = 3 units

Conclusion

In conclusion it is seen that the simulated annealing algorithm achieves convergence when applied to the test case simulation model. The final model that is used for the simulated annealing optimisation method is model SA 2.3. It is seen for this test case simulation model that although the simulated annealing technique produces convergence to a low average cost value, the range that the parameter values can take on is very large.

3.3.4 Nested Partitions

Preliminary Information

Nested Partitions (NP) is a randomised optimisation technique aimed at solving global optimisation problems developed by Shi and Ólafsson [97]. Originally, this method was developed for finite feasible regions but can be extended to problems where the feasible region is either countably infinite or uncountable and bounded. A brief overview of the algorithm is given in Figure 3.14.

The nested partitions algorithm was implemented on the test case simulation model. It is noted that the aim of the nested partitions algorithm is not necessarily to give the optimal

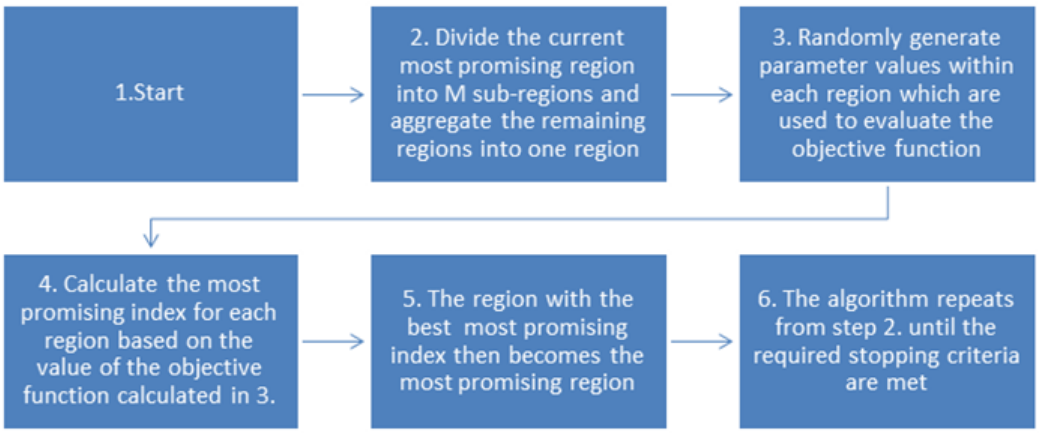

Figure 3.14: Overview of the nested partitions algorithm

parameter values that yield the optimal average cost but rather give a range of the parameter values in which the optimum average cost value lies.

Before the nested partitions algorithm is implemented, the following model specifications need to be decided upon

- The number of partitions created
- The sampling scheme
- The stopping criteria

These are discussed in more detail below.

Again, in the test case simulation model, the two parameters that are to be optimised to minimise the cost are the α and β parameters. Both these parameters have a range of $[0, 1]$. The corresponding feasible region is shown in Figure 3.15.

The number of partitions

The most promising region, which at the start of the algorithm is the whole feasible space, was subdivided into 4 regions. This is done by dividing the range of each parameter by two. This means that after the first iteration of the NP algorithm the feasible region is always divided into 5 partitions - 4 of which are equal size and the remaining partition is the remainder of the feasible region. Due to the fact that there are only two parameters and hence the feasible region is a square the way in which the partitions were calculated is by dividing the most promising region into 4 equally sized squares. Figure 3.16 shows an example of a partitioning. The white square is the most promising region that has been divided into 4 equally sized sub-regions, labelled 1 to 4, and the light blue area represents the 5th partition. The purpose of this 5th partition is to ensure that the algorithm does not converge to a local optimum.

The random sampling scheme

For each sub-region of the most promising region, the parameter values to be used in evaluating the objective function were randomly generated from a uniform distribution that spanned over

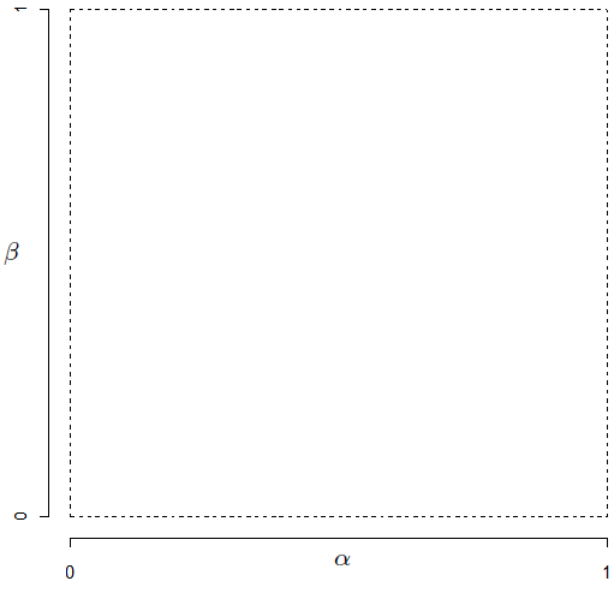

Figure 3.15: Feasible region of the parameters α and β

the range of the parameter for that region. For the remaining region (i.e. the 5th partition), the parameter values were generated from a uniform distribution $[0, 1]$. If the value generated was in one of the sub-regions another random value was generated.

The stopping criteria

For this particular optimisation problem the algorithm was terminated after 12 iterations. This was due to the fact that it was seen that after this number of iterations, there seemed to be no difference in subdividing the most promising regions further. Also, one notes that if the most promising index was found to not belong to a sub-region of the most promising index then the algorithm backtracked. The backtracking consists of beginning the Nested Partitions algorithm again. Although backtracking to the beginning of the algorithm every time that the most promising index is generated from the 5th partition seems rather harsh it is not a hindrance in this test case example as the nested partitions algorithm is quick to run. Should the nested partitions algorithm be run on a more complicated model, the backtracking criteria should be revisited as returning to the beginning every time might not be feasible.

Figure 3.17 shows an example of how the feasible region has been divided. Each colour represents an iteration (therefore there are 12 colours in total). The dots represent the randomly generated α and β pair that were used when running the test case simulation model for each of the regions 1 to 4. The stars represent the randomly generated α and β pair that were used when running the test case simulation model for the 5th partition.

Simulation Optimisation

The nested partitions algorithm is now explained in more detail below:

1. The entire feasible region is divided into 4 equally sized regions

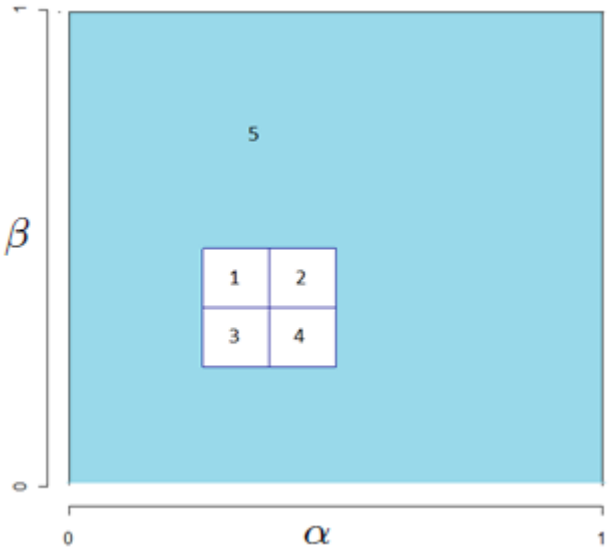

Figure 3.16: Example of partitioning in the nested partitions algorithm

2. In each region, a random parameter value is generated from a uniform distribution from the possible range of the parameter in the region. This gives 4 pairs of α and β values; one pair for each region

3. The test case simulation model is evaluated for each of the pairs of parameter values found in step 2.

4. The new most promising region is chosen to be the region that contains the pair of parameter values that produced the lowest average cost value

5. This most promising region is divided into 4 equally sized regions. The 5th region is the remainder of the feasible space. (An example of this is shown in Figure 3.16)

6. Parameter values for the regions 1 to 4 are randomly generated from a uniform distribution whose range for each parameter corresponds to the parameter's range in that region. The parameter values for the 5th region are generated from a uniform distribution with range $[0, 1]$. If the value generated is in the range of any of the regions 1 to 4 then another parameter value is generated for the 5th partition. In total there are then 5 pairs of α and β values

7. The simulation is run for each pair of values giving a new most promising region

8. This process is repeated from step 4 until it has been iterated 12 times. If however, the most promising region is the 5th partition then the process backtracks to start from step 1 again.

This model takes 6 seconds to run provided that no backtracking has taken place.

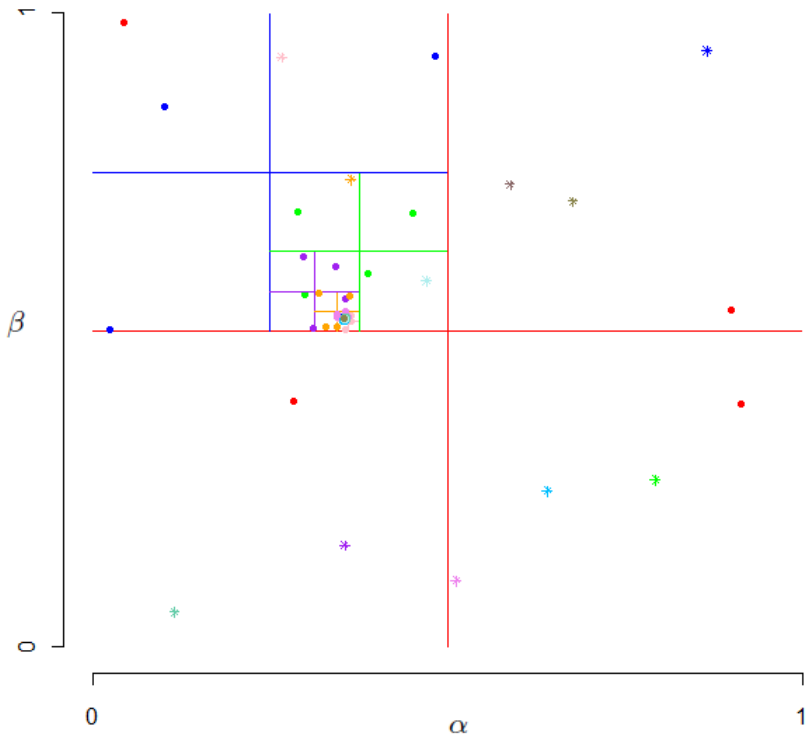

Figure 3.17: Example of a run of the nested partitions algorithm

Final Model The final model specifications are defined as follows:

- Number of partitions: 5 - 4 equally sized partitions and the 5th partition is the remainder of the feasible space
- Number of iterations: 12
- Sampling scheme: Parameter values generated from a uniform distribution

Table 3.14 shows the output generated from 10 runs of the nested partitions algorithm. It is noted that although running the model 10 times is not a lot, it serves as an illustration. The table shows the range of the α and β parameters corresponding to the region in the 12th iteration that gave the most promising index, the randomly generated α and β values from this region and the associated average cost for these parameter values. From these results it can be seen that there are many possible ranges for the α and β parameters which yield similar average cost values. The possible range for each of the parameters is small indicating that convergence is possible for the parameters even though each time the model is run the parameters converge to different values. It is noted that although the average cost value converges to similar values each time the model is run, there are many possible combinations of α and β values that provide similar average cost values.

Run	α range	β range	α	β	average cost
1	[0.325; 0.325]	[0.721; 0.721]	0.325	0.721	1.941
2	[0.842; 0.842]	[0.026; 0.026]	0.842	0.026	1.893
3	[0.207; 0.207]	[0.599; 0.599]	0.207	0.596	1.899
4	[0.928; 0.928]	[0.146; 0.146]	0.928	0.146	1.856
5	[0.494; 0.494]	[0.329; 0.330]	0.494	0.329	1.971
6	[0.738; 0.738]	[0.088; 0.088]	0.738	0.088	1.919
7	[0.652; 0.652]	[0.378; 0.379]	0.652	0.379	1.895
8	[0.104; 0.104]	[0.722; 0.723]	0.104	0.723	2.004
9	[0.245; 0.2450]	[0.627; 0.627]	0.245	0.627	2.028
10	[0.804; 0.804]	[0.059; 0.059]	0.804	0.059	1.905
Mean absolute deviation					0.237

Table 3.14: Output from 10 runs of the final nested partitions algorithm

Changing the storage capacity The same simulation optimisation was now applied to the test case simulation model but this time the storage capacity was increased from 0 to 3 units. Table 3.15 shows the results from 10 runs of this model. From these results it can be seen

Run	α range	β range	α	β	average cost
1	[0.984; 0.984]	[0.130; 0.131]	0.984	0.130	1.121
2	[0.537; 0.537]	[0.467; 0.467]	0.537	0.467	1.084
3	[0.609; 0.609]	[0.534; 0.534]	0.609	0.534	1.115
4	[0.594; 0.594]	[0.244; 0.244]	0.594	0.244	1.030
5	[0.758; 0.758]	[0.351; 0.352]	0.758	0.351	1.112
6	[0.656; 0.657]	[0.199; 0.199]	0.656	0.199	1.053
7	[0.549; 0.549]	[0.453; 0.453]	0.549	0.453	1.134
8	[0.948; 0.949]	[0.136; 0.136]	0.948	0.136	1.070
9	[0.451; 0.451]	[0.521; 0.521]	0.451	0.521	1.319
10	[0.082; 0.083]	[0.866; 0.866]	0.083	0.866	0.992
Mean absolute deviation					0.167

Table 3.15: Output from 10 runs of the final nested partitions algorithm with storage capacity = 3 units

that the model achieved a lower average cost to when there was no storage capacity. This is understandable as less cost is incurred when there is an oversupply as some can be kept in storage and less cost is incurred if there is an under supply as the stock in storage can be used. It is also seen that there is a large number of possible α and β values that yield very similar average cost values.

Conclusion

The nested partitions algorithm works well in finding a minimum average cost when the algorithm is run. However, although the average cost values are very similar when different model runs are compared, the combination of parameter values which generate these average costs are very different. It is noted that using this method, convergence is achieved very quickly. It is noted, however, that the average cost value that is achieved when running the nested partitions method is much higher than that achieved when running the genetic algorithm or simulated

annealing algorithm. This shows that although convergence is achieved fairly quickly with this method, it does come with an associated cost of converging to a higher average cost than the other methods.

3.3.5 Nelson-Matejcik Method (NM method)

Preliminary Information

Prior to defining the NM procedure, Ranking and Selection (R&S) methods and Multiple Comparison Procedures (MCP) are defined. R&S procedures are statistical methods specifically developed to select the best system or a subset that contains the best system design from a set of k competing alternatives. In general, these methods ensure the probability of a correct selection at or above some user specified level. R&S procedures are typically performed in two stages. In the first stage, a set of observations is taken and statistics are used to determine the size of the second stage sample. Indifference zone R&S procedures also allow the specification of an amount, δ, at which they are indifferent between two alternatives [106]. MCPs specify the use of certain pairwise comparisons to make inferences in the form of confidence intervals about relationships among all designs. The most popular MCP, multiple comparisons with the best (MCB) bounds the difference between the expected performance of each system and the best of the others [55]. Like indifference zone R&S, MCB is often performed in two stages. In short, R&S provides the experimenter with the best system design while MCPs provide information about the relationships among the designs (e.g., how much better the best design is in comparison with the alternatives) [106]. To take advantage of the information that both of these methods provide, they were combined by Nelson and Matejcik to form the NM procedure. A brief overview of the method is given in Figure 3.18. A more detailed version of the process is explained in [40].

Simulation Optimisation

The NM procedure was implemented on the test case simulation model following the procedure that was used in [106] where the NM procedure was used to evaluate the design of a family practice healthcare clinic. The process is explained in detail below.

Step One This method is carried out on a sample of the feasible space and not the entire feasible space. Therefore, a sample is needed from the feasible space. An assumption is made such that the feasible space can be sampled uniformly which means that each $\theta \in \Theta$ can be obtained with equal chance by some sampling scheme. It is a statistical fact that under independent uniform sampling a set of representative samples from the feasible space can be obtained [21]. With this assumption in mind, the number of representative samples, N, that is used needs to be decided.

In order to choose N designs from the design space such that at least one of the N samples

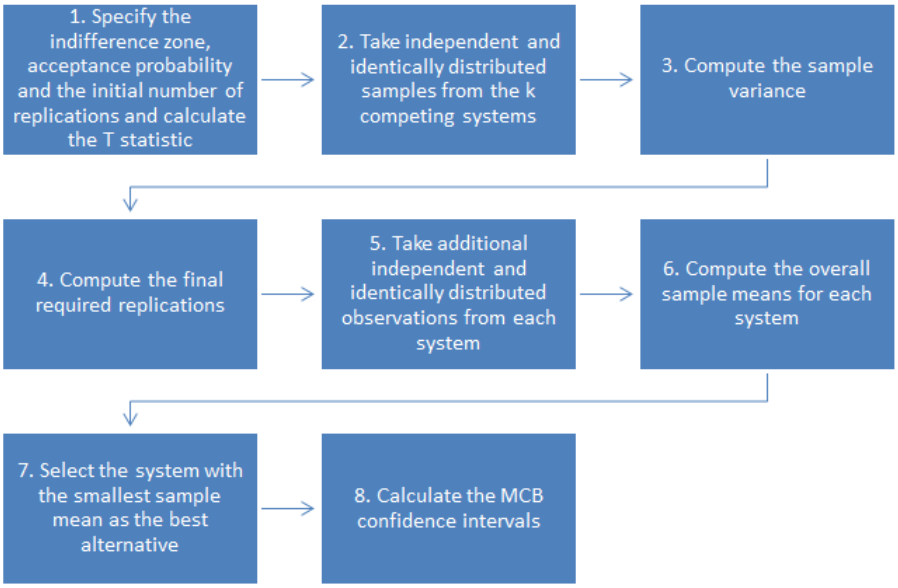

Figure 3.18: Nelson-Matejcik method outline

falls into the top n-percentiles of the feasible space, Θ, the following probability function is used

$$p = 1 - Prob[\text{all N samples not in the top n-percentile of } \Theta] = 1 - (1 - \frac{n}{100})^N$$

For N = 1000 and n = 5, p is calculated as $p = 5.29 \times 10^{-23} \approx 0$. This means that there is a very small chance that none of the 1000 uniformly generated samples belongs to the top 5% of the feasible space. Therefore, 1000 samples are selected and assumed to represent the feasible space. This sample is therefore randomly selected from the feasible space and is defined as $\hat{\Theta}$. The aim of the NM procedure will then be to decide which of these α and β pairs results in the optimum cost value. It is noted that since the NM procedure is performed on a sample of the feasible space that contains at least one design that is in the top 5% of the feasible space, the optimum result generated by this process therefore is defined not as the optimum design but as a design present in the top 5% of designs.

Figure 3.19 plots the 1000 α and β pairs that were randomly selected. From this it can be seen that the feasible space is well represented.

Step Two The next step in the NM procedure is to ensure that the average cost is normally distributed, to ensure that the condition of spherity is satisfied. In order to do that a baseline model is selected and run and the normality tests are performed on the output that is generated. Due to the fact that for both α and β, there is no prior knowledge as to the value that they can take on and any value within their range is equally likely, it is decided to set them both equal to 0.5 for the baseline model and to run the simulation model.

Following the method used in [106], batched means will be used. The method of batched means is frequently used to estimate the steady-state mean and/or variance of a simulation

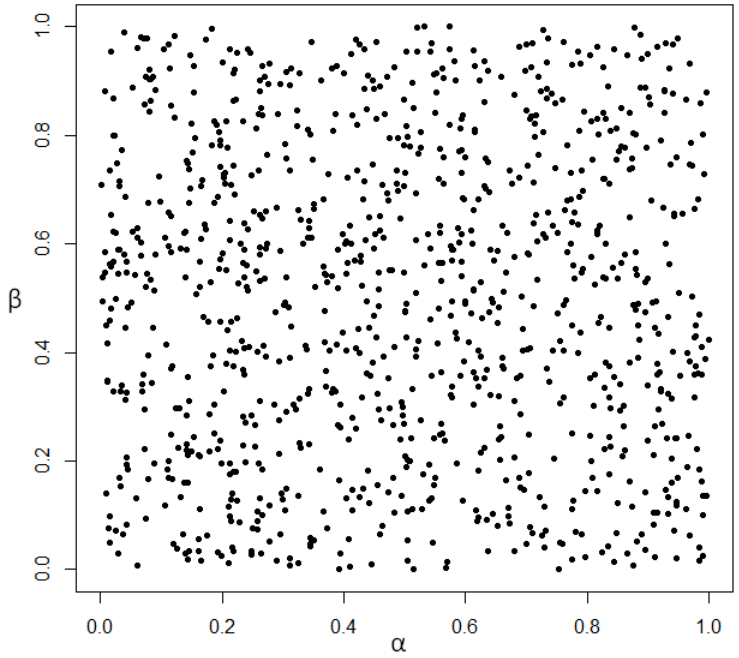

Figure 3.19: Sample of α and β pairs from feasible region

output measure, in this case average cost [3]. Batched means is an attractive technique because it allows the simulation analyst to avoid making an excessive number of replications, each of which must discard data due to initialisation bias, to obtain a valid steady-state mean and/or variance point estimator [106]. The standard approach to the method of batched means is to divide the steady-state output from a single replication into contiguous batches whose sample means are treated as independent observations. Given an output series $X_1, X_2, ..., X_N$ of a simulation run, m batches of size $b = n/m$ are formed. It is noted that there is much debate as to the correct number of batches. According to Schmeiser [91] it has been seen that using between 10 to 30 batches is reasonable. For more information on this justification see [91]. For the purposes of this case, the first 100 average cost values are dismissed (as the process was still reaching steady-state) and 30 batches of size 30 are then created based on the remaining 900 average cost values generated from the simulation model.

The baseline model was then run for a length of 1000 iterations of the simulation model and at the end of each iteration the average cost values were collected. This means that at the end of the simulation run there were 1000 cost values. Of these 1000 values, the first 100 were discarded and the remaining 900 were batched into 30 batches of size 30 each. The simulation model was run to have a total of 30 replications (the number 30 was chosen as the initial number of model runs) and the batched values were calculated each time.This means that there was now 30 values for each batch. For each batch, the average of the 30 values was taken and is shown in Figure 3.20. Three tests of normality were then applied to these 30 batched means: Anderson-Darling, Kolmogorov-Smirnov and Shapiro-Wilk. The results are shown in table 3.16. The large p-values associated with the tests suggest that the null hypothesis of normally distributed data

would not be rejected at any reasonable confidence level. This suggests that it is reasonable to proceed with the NM procedure.

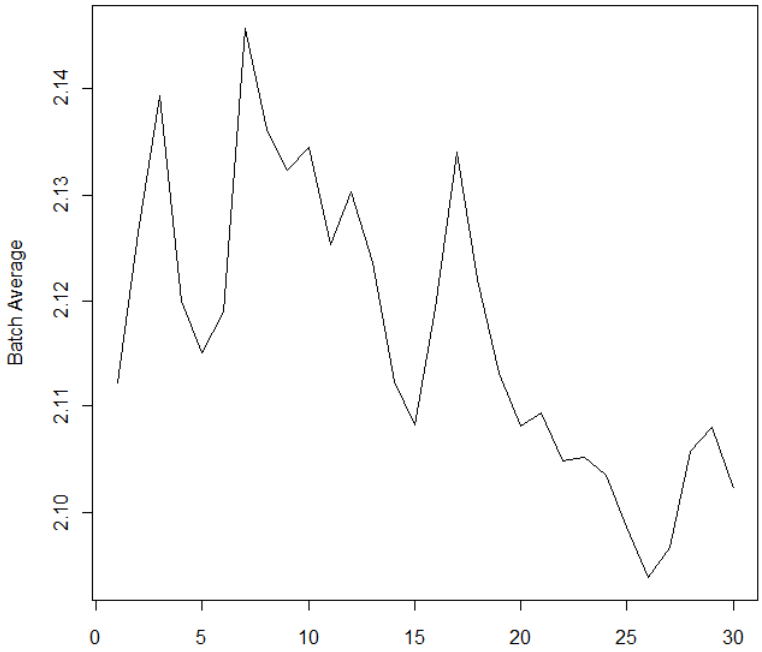

Figure 3.20: Batched average values for thirty iterations of the baseline simulation model

Normality Test	**p-value**
Anderson-Darling	0.476
Kolmogorov-Smirnov	0.503
Shapiro-Wilk	0.522

Table 3.16: Comparison of tests for normality for Nelson-Matejcik method

Step Three The next step in the process is to create a confidence level on the output generated from running the simulation model for the 1000 α and β pairs in order to select a sample to proceed with. In order to do this, the simulation model was run once for each of the α and β pairs and the output was recorded. A 95% confidence level on the mean output value from the 1000 runs was calculated as [2.265; 2.303]. From this, the designs that had an average cost of less that 2.303 were selected as the sample on which further analysis will be performed. The sample size decreased to 603 α and β pairs. Figure 3.21 plots these 603 α and β pairs. From this graph, it appears as though α and β pairs in which both values are less than 0.4 of where both values are larger than 0.8 do not yeild optimal solutions. This comfirms what the output graph (Figure 3.5) showed. The reason that this step is carried out it to reduce the size of Θ by eliminating solutions that are not optimal so that the method can continue with a smaller sample size and therefore complete in a shorter amount of time. The reason that the solutions that are lower than the top bound of the confidence interval are kept is since there is simulation

noise there is a chance that the optimal solution did not perform in the best way on this run of the model. By keeping this group and running further the model more times below, this simulation noise will be accounted for and thereby help identify the optimum solution of the sample.

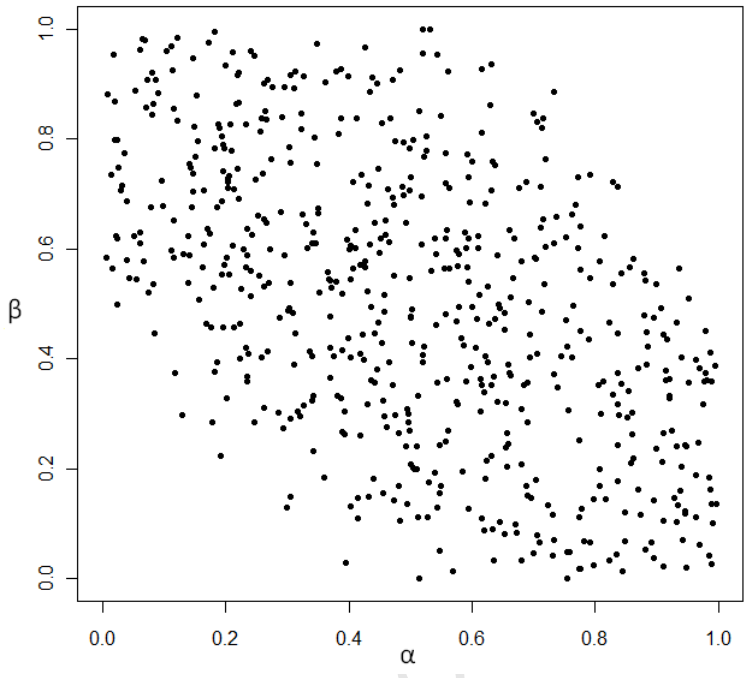

Figure 3.21: Selected Sample From Feasible Region

Step Four Now that the final sample has been selected, step one from the NM outline can be performed. The following values are specified:

- Indifference zone, $\delta = 0.05$. This means that designs that create outputs differ but less than 0.05 are indifferent to one another.
- α value $= 0.05$. This means that the probability of correctly selecting the optimum design is 0.95.
- Number of observations to be collected during the first stage of the procedure $= 30$. The reason for this choice of number of initial observations is due to the fact that it has been used in literature [106, 105].

The next value that is calculated is g where $g = T^{1-\alpha}_{k-1,(k-1)(n_0-1),0.50}$ is the $(1-\alpha)$-quantile of the maximum of a multivariate t random variable with $k-1$ dimensions, $(k-1)(n_0-1)$ degrees of freedom and common correlation 0.50. This is obtained through the tables in [52] and is equal to 2.68.

Step Five Now that the initial number of replications has been decided, each of the 603 designs are run 30 times and the output is captured. It is noted that the technique of common random numbers is used for this process. This means that the same random numbers are used for the same replication across all the designs.

From this output, the sample variance is then calculated. The sample variance is defined as

$$S^2 = \frac{2\sum_{i=1}^{k}\sum_{j=1}^{n_0}(Y_{ij} - \bar{Y}_{i\cdot} - \bar{Y}_{\cdot j} + \bar{Y}_{\cdot\cdot})^2}{(k-1)(n_0-1)}$$

where Y_{ij} is the j^{th} replication of the i^{th} design and $\cdot$ represents averaging with respect to that subscript. The sample variance for the 30 replications of the 603 designs is equal to 0.0625.

Step Six This next step is to calculate the final number of replications required for each design as shown below.

$$\begin{aligned} N &= max[n_0, (gS/\delta)^2] \\ &= max[30, 179] \\ &= 179 \end{aligned}$$

This means that an extra 149 replications of each of the designs is needed. These 149 replications for each of the 603 designs are performed and the results are captured. Common random numbers are also used for these replications.

Step Seven Now that all 179 replications of the 603 designs have been generated, the overall sample means for each design are calculated as follows:

$$\bar{Y}_{i\cdot} = \frac{1}{N}\sum_{j=1}^{N} Y_{ij} \text{ for } i = 1, 2, ..., k$$

The design that has the smallest overall sample mean is selected as the best design. The specifications for this model is shown in table 3.17. From a R&S perspective, this means that with probability greater than or equal to 0.95 that this design, that has a mean average cost of 2.05, lies within 0.05 of the true best design.

Mean average cost	2.05
α	0.85
β	0.07

Table 3.17: Specification of best design for Nelson-Matejcik method

Step Eight The last step of the process is to calculate the confidence intervals. This is done to see if the output value from the other designs are within the indifference zone. The confidence

intervals are calculated using the following formula:

$$\mu_i - \min_{j \neq i} \mu_j \in [-(\bar{Y}_{i\cdot} - \min_{j \neq i} \bar{Y}_{j\cdot} - \delta)^-, (\bar{Y}_{i\cdot} - \min_{j \neq i} \bar{Y}_{j\cdot} + \delta)^+] \text{ for } i = 1, 2, ..., k$$

where $-x^- = min[0, x]$, $x^+ = max[0, x]$ and μ_i is the mean of the 179 runs of design i.

Examination of the MCB confidence intervals provides inferences on the (assumed) superiority of the best design identified above. After having calculated the confidence intervals, 90 other designs have an MCB interval that contain 0. This means that from an MCB perspective, there is no one uniformly superior design. These 90 designs are all superior to the remaining systems, however, there is no clear winner among them. These designs are plotted in Figure 3.22. The red point represents the design that was selected as the best design. Also, Table 3.18 shows the confidence intervals for 10 of the 90 designs that produced the lower average cost. The mean absolute deviation for these designs is equal to 0.210. It is noted that although none of the 90 designs are superior, a selection of 10 was done to illustrate the confidence intervals. It can be seen that all of these confidence levels contain 0.

Design	α	β	average cost	lower MCB limit	$\bar{Y}_{i\cdot} - \min_{j \neq i} \bar{Y}_{j\cdot}$	upper MCB limit
1	0.848	0.068	2.052	-0.051	-0.001	0.049
2	0.895	0.037	2.053	-0.049	0.001	0.051
3	0.874	0.116	2.056	-0.046	0.004	0.054
4	0.835	0.044	2.062	-0.039	0.011	0.061
5	0.882	0.052	2.063	-0.039	0.011	0.061
6	0.688	0.169	2.063	-0.039	0.011	0.061
7	0.911	0.023	2.064	-0.038	0.012	0.062
8	0.774	0.018	2.065	-0.036	0.014	0.064
9	0.695	0.146	2.066	-0.036	0.014	0.064
10	0.844	0.014	2.066	-0.036	0.014	0.064

Table 3.18: Confidence interval for a selection of 10 designs from the 90 best designs for Nelson-Matejcik method

Changing the storage capacity The NM procedure was then repeated using the same steps listed above when the simulation model was run with a storage capacity. The storage capacity, similarly to the other procedures, was set to be equal to 3 units.

Firstly, the 1000 randomly selected possible solutions are run through the simulation model once to get an output value. A 95% confidence interval was then built on the output values and was defined as [1.534; 1.593]. Again, since the problem is a minimisation one, all models that produced an output value that was less that 1.593 were selected. This led to the sample of models decreasing from 1000 models to 645 models. The α and β pairs for these 645 models are plotted in Figure 3.23. Again, it can be seen that α and β pairs where both the values are larger than 0.8 or both the values are less than 0.4 are excluded from the sample.

The same indifference level of 0.05 as well as the probability of correctly selecting the optimum design is 0.95 and the number of observations to be collected during the first stage of the procedure is 30 are kept. Similarly the value of g is calculated where $g = T^{1-\alpha}_{k-1,(k-1)(n_0-1),0.50}$ is

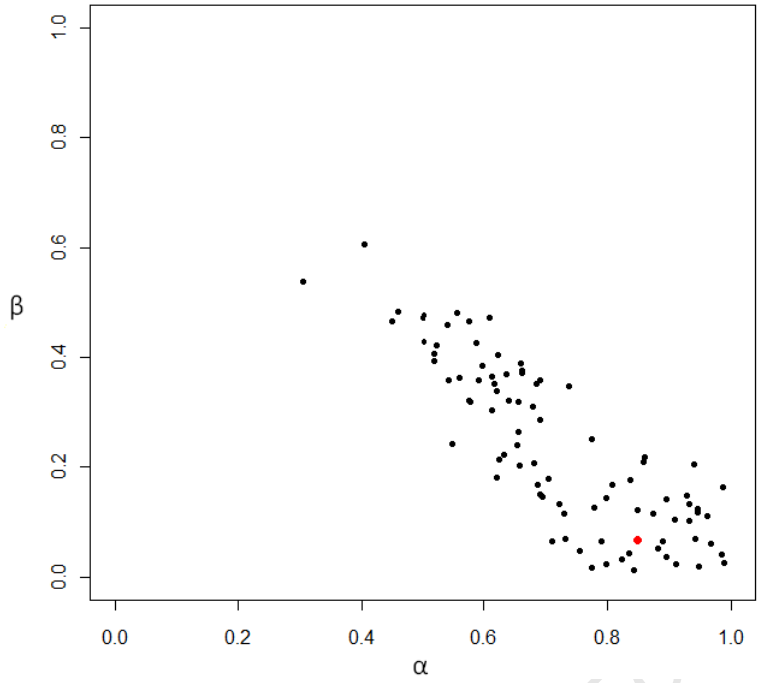

Figure 3.22: Superior designs generated from Nelson-Matejcik method

the $(1-\alpha)$-quantile of the maximum of a multivariate t random variable with $k-1$ dimensions, $(k-1)(n_0-1)$ degrees of freedom and common correlation 0.50. This is obtained through the tables in [52] and is equal to 2.68.

The 645 models selected were then run 30 times. From the output values, the sample variance, S^2, was then calculated and was equal to 0.054888. From this, the final number of replications required for each design was calculated as 158. This meant that an extra 128 replications of the design were required.

Once all 158 replications for each of the 645 designs were run, the overall sample mean for each design was calculated. The design with the smallest overall sample mean was selected as the best design and the specifications for this model are shown in table 3.19

Mean average cost	1.133
α	0.778
β	0.127

Table 3.19: Specification of best design when storage capacity = 3 units for Nelson-Matejcik method

Lastly, the MCB confidence intervals were calculated. From these intervals, it was seen that 53 other designs have an MCB interval that contains 0. This means that from an MCB perspective, there is no one uniformly superior design. These 53 designs are all superior to the remaining systems, however, there is no clear winner among them. These designs are plotted in Figure 3.24. The red point represents the design that was selected as the best design.

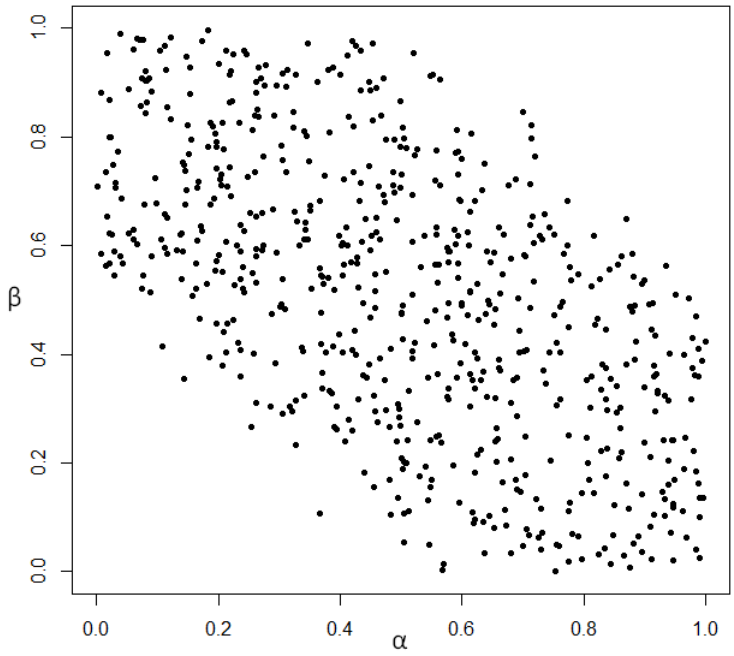

Figure 3.23: Selected sample from feasible region for model with storage capacity for Nelson-Matejcik method

Conclusion

From the implementation of the NM method, an optimum design was found as well as 89 designs that produced results within the indifference zone defined. The fact that 90 designs were found to be indifferent means that there are many pairs of α and β values that produce very similar average cost values. When a storage capacity was included in the simulation model, 53 designs were found to be indifferent and produce very similar average cost values. It is interesting to notice that for both the model with no storage capacity and the model with storage capacity, similar optimum solutions that yielded minimum average cost were found.

The results achieved by this method however are not good. This is seen through the fact that the nested partitions, genetic algorithm and simulated annealing techniques produce much better results, in terms of a lower average cost, than those achieved with the NM method.

3.3.6 Ordinal Optimisation

Preliminary Information

Ordinal optimisation (OO) is an important tool used to deal with simulation optimisation. If one does not insist on getting the optimal design i.e. the goal is softened by having a high probability of getting any good enough design, the problem will become more approachable. When the goal is softened, one can tolerate imprecise performance estimates because there is high confidence in obtaining a good enough design from the selected set [51]. There are, therefore, two main ideas with OO. These are that 'order' is much more robust against noise

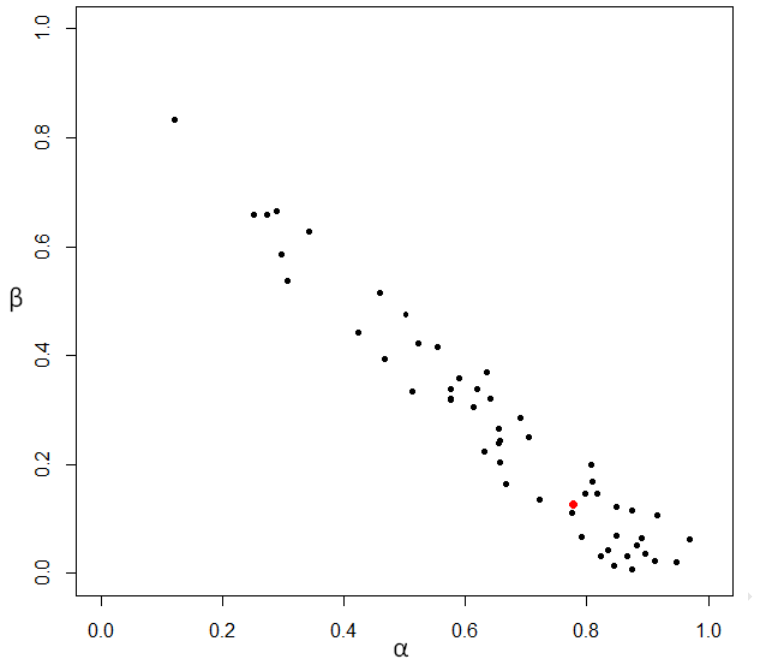

Figure 3.24: Superior designs for simulation model with storage capacity for Nelson-Matejcik method

than 'value' and this method does not insist on getting the 'best' but is willing to settle for good enough. This type of ordering of the different designs in the design space can usually be done using a very crude and hence computationally simple model of the performance of the systems [58]. Using a crude model can mean a very short simulation or averaging over a small number (even one) of sample paths. For the purposes of this test case simulation model, the crude simulation model is the same as the test case simulation model except that it is run for a shorter simulation length.

The procedure for the practical application of OO to simulation optimisation problems is defined as follows [56]: Let Θ be the search space of the optimisation variables, Θ^N the set of N chosen designs, N the number of designs uniformly chosen from Θ, G the good enough subset, usually the the top g designs in Θ^N as defined by the by the accurate simulation model, S the selected subset, usually the top s designs in Θ^N as defined by the crude simulation model, $G \cap S$ the set of truly good enough designs in S, AP the alignment probability $= Pr[|G \cap S| \geq k]$, the probability that there are actually k truly good designs in S and G. These definitions and concepts are shown graphically in Figure 3.25.

Step 1 Uniformly sample N designs from Θ to form Θ^N.

Step 2 Estimate the performance of the designs in Θ^N using a crude and computationally fast model.

Step 3 Specify the size of the good enough subset, g, the required alignment level, k, the corresponding alignment probability, AP, and the error bound.

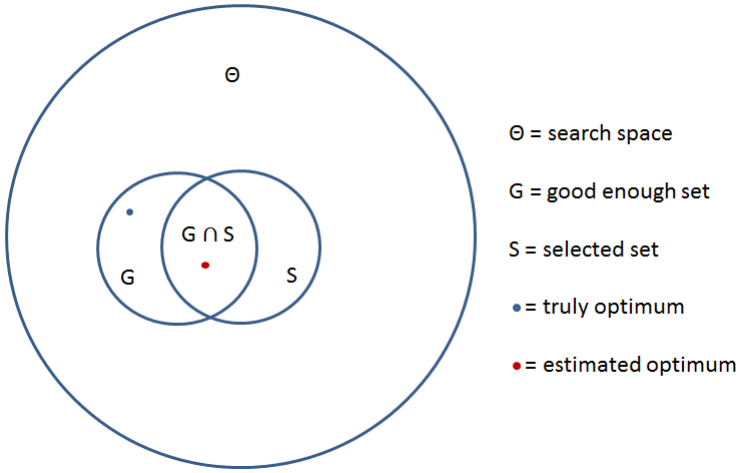

Figure 3.25: Ordinal optimisation method represented graphically

Step 4 Determine the size of the selected subset, s.

Step 5 Select the estimated top s designs from step two to form the selected subset, S.

Step 6 OO theory ensures that S contains at least k truly good enough designs with a probability level no less than AP.

The above process is outlined in Figure 3.26. It is noted that because of the efficient narrowing down process promised by OO, a designer can learn from one-step application of OO the properties of good solutions, such as which portion of the design space is to be explored next or what a better representation of the problem should be [65].

Ordinal Optimisation Implementation

The OO process is now implemented on the test case simulation model and is explained below. The aim of this method is to get a group of sample designs that contain k truly good enough designs with a probability level no less than the defined alignment probability.

Step One In the simulation optimisation model, the aim is to minimise the average cost through ordering the correct amount of fuel which is decided through the selection of the values of α and β. For the purposes of this test case simulation model, α and β are continuous variables and can take on any value between $[0, 1]$. This means that there are many possible combinations of α and β. As a result, instead of testing all these pairs, a sample of α and β pairs are taken from the feasible space. Using the same justification used in the NM procedure, the number of samples taken from the feasible space is 1000.

Step Two The next step is to estimate the performance of these 1000 selected designs using a crude and computationally fast model. The test case model that we are using is already very

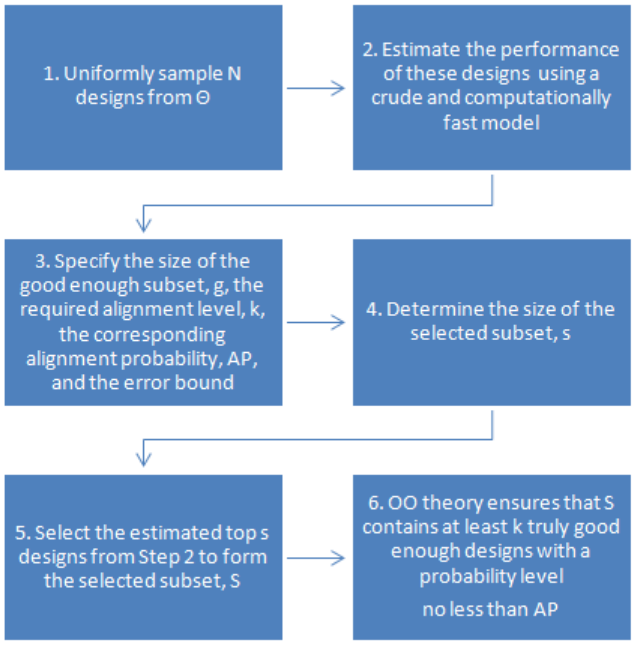

Figure 3.26: Ordinal optimisation method

simple and computationally fast but to illustrate the OO procedure correctly the crude model is chosen to be the same as the test case simulation model except the number of iterations has been cut down from 1000 iterations to 300 iterations. As previously mentioned when explaining the test case simulation model, the simulation model reaches steady state between the 150^{th} and 200^{th} iteration and therefore decreasing the number of iterations to 300 does not mean that the simulation model is stopped before steady state has been reached.

The time it takes to run one simulation model with 1000 iterations is 0.5 of a second. When the simulation model is then run with 300 iterations, it takes 0.17 of a second. This means that the running time has decreased by 67% when decreasing the number of iterations from 1000 to 300.

The 1000 samples selected in step one are then run using this crude model and the output is recorded. From the output generated, an ordered performance curve (OPC) is drawn. The OPC is constructed based on the estimated performance values obtained by the crude model. The 1000 estimated performance values are arranged in an ascending order (since this is a minimization problem). The x axis of the resulting plot is the scheme labels; whereas the y axis represents the (estimated) performance values. The shape of an OPC determines the nature of the underlying optimisation problem. The shapes of OPC curves can be broadly categorized into five categories: flat, u-shape, neutral, bell and steep, as shown in Figure 3.27 [65, 113].

For a minimization problem, a smaller performance value means a good scheme, and a larger performance value means a bad scheme. For a problem, if more small-value schemes are found, then the problem has more good schemes. The five OPC curves in Figure 3.27 represent five classes of optimisation problems: flat shape OPC has many good schemes, a U-shaped OPC

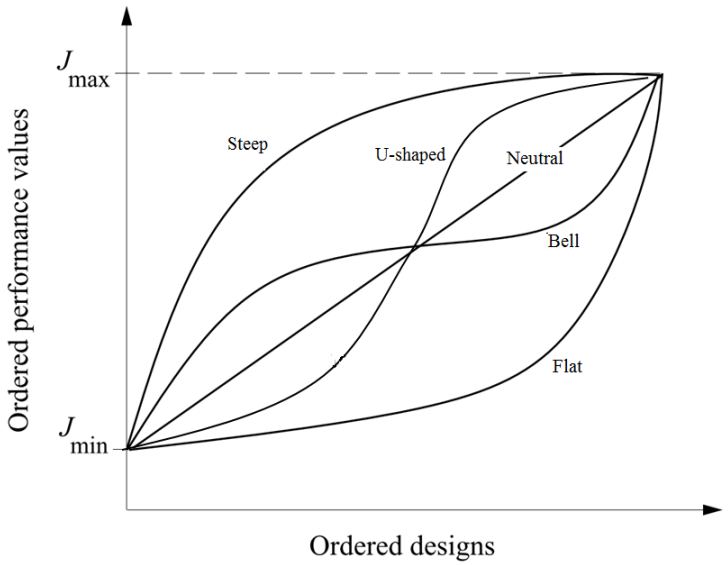

Figure 3.27: Five different types of Ordered Performance Curves for ordinal optimisation [65, 113]

has many good and bad schemes, a neutral shape OPC has good and bad schemes equally distributed, a bell shape OPC has many intermediate schemes and lastly, a steep shape OPC has many bad schemes.

The OPC derived from the output of the 1000 samples is shown in Figure 3.28. From this it can be seen that the OPC shape is bell shaped which means that there are many intermediate schemes. It is also noted that there are very few designs which have very high output values and it seems as though towards the higher ordered designs, the OPC curve almost resembles a flat shaped OPC.

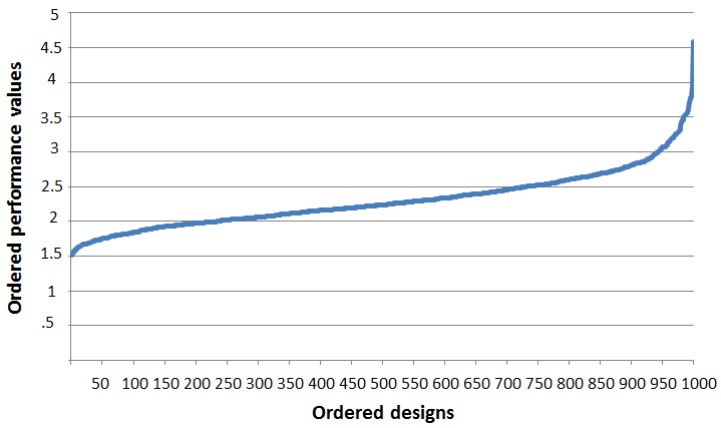

Figure 3.28: Ordered Performance Curves of output from crude model for ordinal optimisation

Step Three The size of the good enough subset, the required alignment level, the corresponding alignment probability and the error bound are then defined as follows:

- The good enough subset G is defined as the top 5% solutions of Θ^N, thus $g = N \times n\% = 1000 \times 5\% = 50$.
- The alignment level $G \cap S$ is $k = 1, 2, 3, 4$ or 5.
- The alignment probability of $G \cap S$ is $p = 95\%$.
- The error bound is then calculated as described below using the method described in [43]. The error bound is necessary in order to calculate the size of the selected subset, the method of which is explained briefly in step four.
  - 30 samples are randomly selected from the 1000 samples. A total of 30 samples is chosen as it has been shown as a good first stage sample size [113].
  - These 30 samples are run using the test case simulation model and their output is recorded.
  - From the output, the variance of the normalised approximation error is calculated to be 0.168.
  - The error bound, W, which is used to define the distribution $U[-W, W]$ is then calculated as

$$\text{variance of the uniform distribution} = 0.168 = \frac{1}{12}(2W)^2$$

$$W = 0.507$$

The value is rounded to be equal to 0.5.

Step Four In [65], through extensive simulations and statistical analysis, a formula is derived to relate the size of the selected subset S to the shape of the OPC, the size of the good enough subset G, the alignment level k, the alignment probability p and the error bound - all of which are quantities that have been decided or derived in the previous steps and are now summarised:

- Size of the representative subset $N = 1000$.
- The OPC is bell shaped.
- The good enough subset G is defined as the top 5% solutions of Θ^N, thus $g = N \times n\% = 1000 \times 5\% = 50$.
- The alignment level $G \cap S$ is $k = 1, 2, 3, 4$ or 5.
- The alignment probability of $G \cap S$ is $p = 95\%$.

- The error bound is defined to be 0.5. (It is noted that for the derivation of this formula, three values of W are considered: $W = 0.5, 1.0, 2.5$ where the smaller the value of W, the less amount of noise.)

The formula used to calculate the required subset size is defined as

$$Z = Z(g, k, N, C, \xi(\cdot), P) = exp^{Z_0} k^{\rho} g^{\gamma} + \eta$$

where g is the size of the good enough subset, k is the number of truly good designs, N is the number of sample designs, C is one of the 5 classes of the OPC, $\xi(\cdot)$ is the noise characteristic generated from U $[-W, W]$ and P is the acceptance probability. Also, Z_0, ρ, γ and η are constants depending on the OPC type and noise characteristics. It is noted that this function is decreasing in g and increasing in k. For a detailed explanation as to the derivation of this formula see [65].

The size of the selected subset for the optimisation problems with the five different OPC shapes are calculated based on the above formula and the results are tabulated in Table 3.20. Since the OPC for this problem is bell shaped the size of the selected subset for the various values of k are in bold in the table. Suppose we are looking at the case where $k = 1$. The

	OPC shape				
g = 50	Flat	U-shaped	Neutral	**Bell**	Steep
k=1	37	25	22	**12**	11
k=2	63	41	35	**15**	13
k=3	88	57	48	**21**	14
k=4	113	73	61	**29**	16
k=5	136	89	71	**39**	19

Table 3.20: Size of selected subset for five OPC based problems for ordinal optimisation

size of the selected subset is then $s = 12$. This means that, after a rough estimation, if the 12 best schemes are picked from the rough estimation to run the exact evaluations, there is a 95% probability that at least one scheme out of the 12 will fall in the good enough subset G. If the alignment level is set to be 2, then 15 schemes need to be calculated for exact evaluations to guarantee that at least two schemes will fall in the good enough subset G with the probability of 95%. As explained in [113], from the rows of table 3.20, the size of the selected subset decreases from a flat shape to a steep shape. This means that an optimisation problem with a flat OPC shape requires more selected schemes than a problem with a steep OPC curve. This is because for a flat shape OPC, there are many schemes whose performance values based on the crude model are more or less equally good (small). The exact performance value of each individual scheme is the estimated value plus the error in the estimation. The error terms are unknown. Adding such error terms will change the order of the ranking of the schemes that are flat (more or less the same performance) significantly. Therefore, more schemes need to be selected in order to capture enough good enough schemes. On the other hand, if the OPC is steep, although fewer schemes out of the 1000 are needed for an exact calculation, the quality of the 1000 samples may be lower. In this case, it may be prudent to increase the initial sample

from 1000 to a larger number in order to capture more 'good enough' schemes.

Step Five and Six The size, s, of the selected subset for the various values of k has now been determined. The top s designs for each value of k are then selected (based on the output given by running the samples through the crude model) and these designs are run using the test case simulation model and the output is recorded. From this, it can be seen that the design that yields the lowest average cost in this particular case has an α value of 0.7259 and a β value of 0.06337 and an average cost value of 1.72812 units. This design was ranked third when producing the OPC using the crude model. We now have selected subsets that have at least k designs in G with a probability of 95%.

Validation of Process The validation of the OO approach and its corresponding results requires the good enough subset be calculated. This is done by running all the 1000 samples using the test case simulation model and selecting the top 50 designs. As already explained, if alignment level is set to be k, then of the s designs making up the selected subset S, at least k schemes will fall in the good enough subset G with the probability of 95%. Figure 3.29 plots the α and β values of the good enough subset and the selected subset for the various values of k. From this, the intersection of the good enough subset and the selected subset are shown. It can be seen that for all values of k, there are more designs in both S and G than required. This is mainly due to the fact that the crude simulation model and the accurate simulation model are very simular. Although many of these designs fall in both S and G, the ordered ranking of these designs are different. This is due to the fact that there is inherent randomness in the test case simulation model and therefore, each time the model is run, a different output can be generated. Table 3.21 illustrates this by showing the ordered ranking of the good enough subset and of the selected subset for the case where $k = 1$. The numbers represent the design scheme (this can range between 1 and 1000 as there are 1000 samples). The design schemes in bold represent the designs in both S and G. From this it can be seen that three designs are in both S and G. In both subsets, design 651 performs the best. As for designs 144 and 856, design 144 performs better than 856 for the selected subset and design 856 performs better than design 144 for the good enough subset. It can also be seen that both the good enough subset designs as well as the selected subset designs for the various values of k have an output values (average costs) that are in keeping with the what is to be expected based on Figure 3.5, which shows the output for various values of α and β.

Table 3.22 shows the α, β and average cost values (generated when calculating the good enough subset) of the designs that are in both S and G for the various values of k.

It is noted that should no model validation take place, there would be no way of knowing which designs in S are also in G.

Implementation of OO with a storage capacity included

The OO method is now implemented on the test case simulation model except that now the storage capacity is increased from 0 units to 3 units (again, this storage capacity is arbitrarily

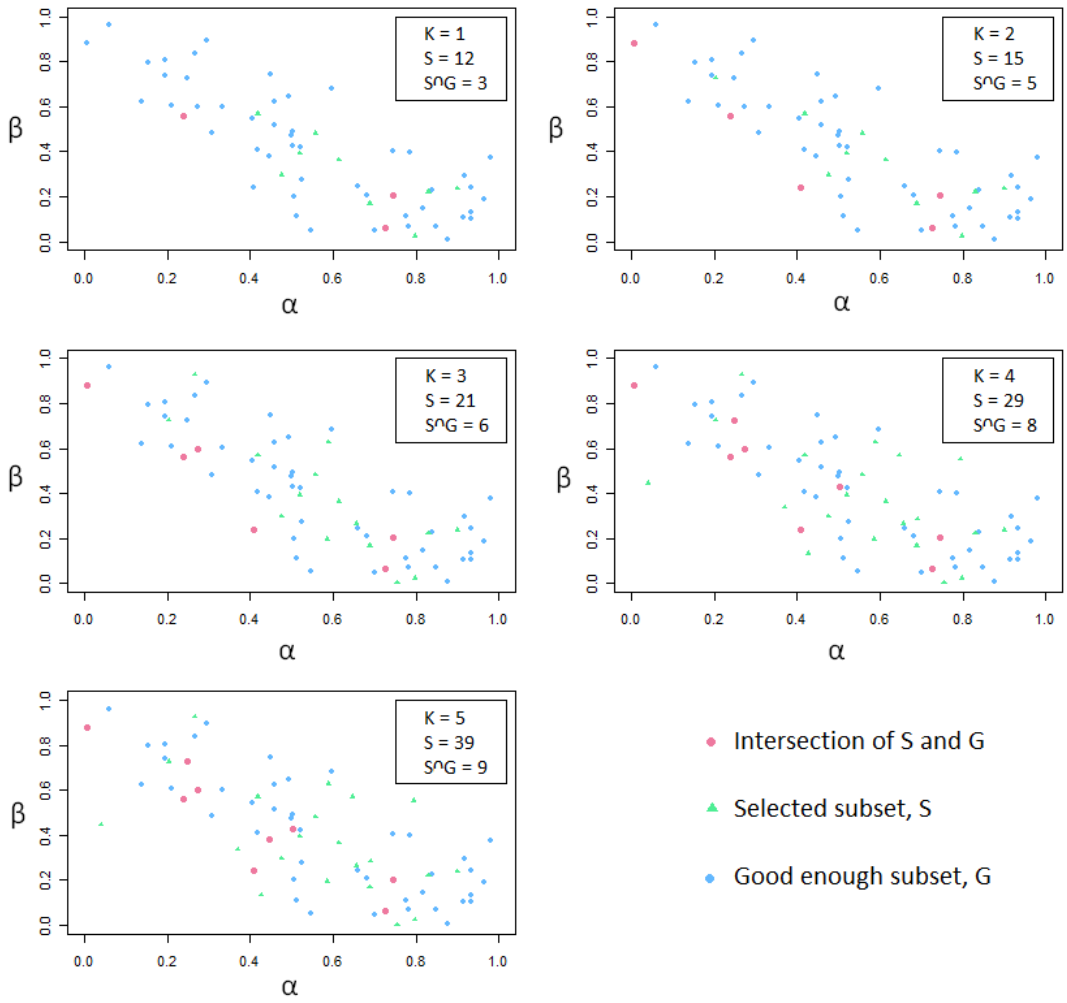

Figure 3.29: α and β values of the good enough subset and the selected subset for the various values of k for ordinal optimisation

chosen). The same process outlined in Figure 3.26 is used and as a result, will not be discussed in as much detail.

Step One The sample designs that are used for this implementation of OO are the same 1000 sample designs that are used when implementing OO when there is no storage capacity included.

Step Two The 1000 sample designs are then put through the crude simulation model. This crude simulation model is the same as the test case simulation model except that instead of being iterated 1000 times, it is only iterated 300 times. From the output generated (which represents the average cost), the values are ordered from smallest to largest and the OPC is drawn as seen in Figure 3.30. From this it can be seen that the graph takes on a bell shape which means that there are many intermediate solutions.

Selected subset (s = 12)	**651**, **144**, 198, 154, 434, 29, 358, 427, 85, 858, 6, **856**
Good enough subset (g = 50)	172, 108, 106, 119, 132, 932, **651**, 239, 328, 513, 540, 340, 552, 703, 876, 295, 890, 473, 996, 326, 115, 32, **856**, 509, 272, 789, 677, 3, 320, 27, 244, 650, 690, 662, 915, 284, 467, 114, 410, 868, **144**, 413, 156, 818, 584, 707, 9, 231, 134, 501

Table 3.21: Ordered designs for the selected subset, when k=1, and for the good enough subset for ordinal optimisation

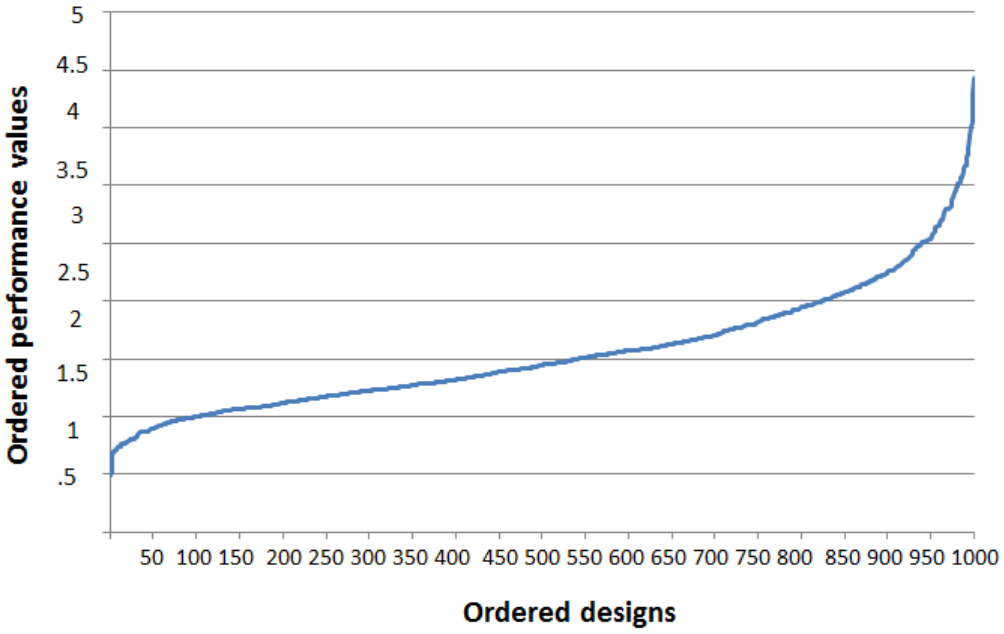

Figure 3.30: Ordered Performance Curve of output from crude model with storage capacity included for ordinal optimisation

Step Three The size of the good enough subset, the required alignment level, the corresponding alignment probability and the error bound are then defined as follows:

- The good enough subset G is defined as the top 5% solutions of Θ^N, thus $g = N \times n\% = 1000 \times 5\% = 50$.
- The alignment level $G \cap S$ is $k = 1, 2, 3, 4$ or 5.
- The alignment probability of $G \cap S$ is $p = 95\%$.
- The error bound is then calculated, using the same method as above, and is defined as $W = 0.6125842$. This is rounded down to be equal to 0.5. This is due to the fact that there are three possible values that W can take on when deciding on the size of the subset selection:0.5, 1 and 1.5. Since the error bound is closest in size to the value of 0.5, it is rounded down to this.

k=1			k=2			k=3			k=4			k=5		
α	β	average cost	α	β	average cost	α	β	average cost	α	β	average cost	α	β	average cost
0.725	0.063	1.789	0.725	0.063	1.789	0.725	0.063	1.789	0.725	0.063	1.789	0.725	0.063	1.789
0.240	0.560	1.906	0.240	0.560	1.906	0.240	0.560	1.906	0.240	0.560	1.906	0.240	0.560	1.906
0.745	0.203	1.862	0.745	0.203	1.862	0.745	0.203	1.862	0.745	0.203	1.862	0.745	0.203	1.862
			0.007	0.880	1.877	0.007	0.880	1.8772	0.007	0.880	1.8772	0.007	0.880	1.8772
			0.406	0.240	1.912	0.406	0.240	1.912	0.406	0.240	1.912	0.406	0.240	1.912
						0.272	0.599	1.863	0.272	0.599	1.862	0.272	0.599	1.862
									0.248	0.725	1.821	0.248	0.725	1.821
									0.502	0.427	1.865	0.502	0.427	1.865
												0.446	0.380	1.823
MAD		0.042	MAD		0.035	MAD		0.03	MAD		0.028	MAD		0.031

Table 3.22: Parameter and output values for the selected subsets with varying values of k for ordinal optimisation as well as the mean absolute deviation (MAS

Step Four The size of the selected subset is now defined using the following characteristics and table 3.20:

- Size of the representative subset $N = 1000$.
- The OPC is bell shaped.
- The good enough subset G is defined as the top 5% solutions of Θ^N, thus $g = N \times n\% = 1000 \times 5\% = 50$.
- The alignment level $G \cap S$ is $k = 1, 2, 3, 4$ or 5.
- The alignment probability of $G \cap S$ is $p = 95\%$.
- The error bound is defined to be 0.5.

This means that the size of the selected subset is 12, 15, 21, 29 and 39 for the corresponding values of k = 1, 2, 3, 4 and 5.

Step Five and Six The size, s, of the selected subset for the various values of k has now been determined. The top s designs for each value of k are then selected (based on the output given by running the samples through the crude model) and these designs are run using the test case simulation model and the output is recorded. From this, it can be seen that the design that yields the lowest average cost in this particular case has an α value of 0.3894 and a β value of 0.5190 and an average cost value of 0.9504 units. This design was ranked 32nd when producing the OPC using the crude model. We now have selected subsets that have at least k designs in G with a probability of 95%.

Validation of Process The OO process is the validated in the same way as above. Figure 3.31 plots the α and β values of the good enough subset and the selected subset for the various values of k. From this, the intersection of the good enough subset and the selected subset are shown. It can be seen that for all values of k, there are more designs in both S and G than required. Although many of these designs fall in both S and G, the ordered ranking of these designs are different. This is due to the fact that there is inherent randomness in the test case simulation model and therefore, each time the model is run, a different output can be generated. It is interesting to note that although the good enough subset has parameter values as one would expect based on what was seen in Figure 3.5, the selected subset, as well as some of the intersection of the selected and good enough subset, has parameter values that don't produce a minimum average cost. This indicates that should the OO method be used without validation, there output that the parameter values would generate would be very varied.

Conclusion

Through this process, we now have sample designs which fall into a good enough subset. Due to the nature of OO and its goal softening property, we are not able to quantify the difference

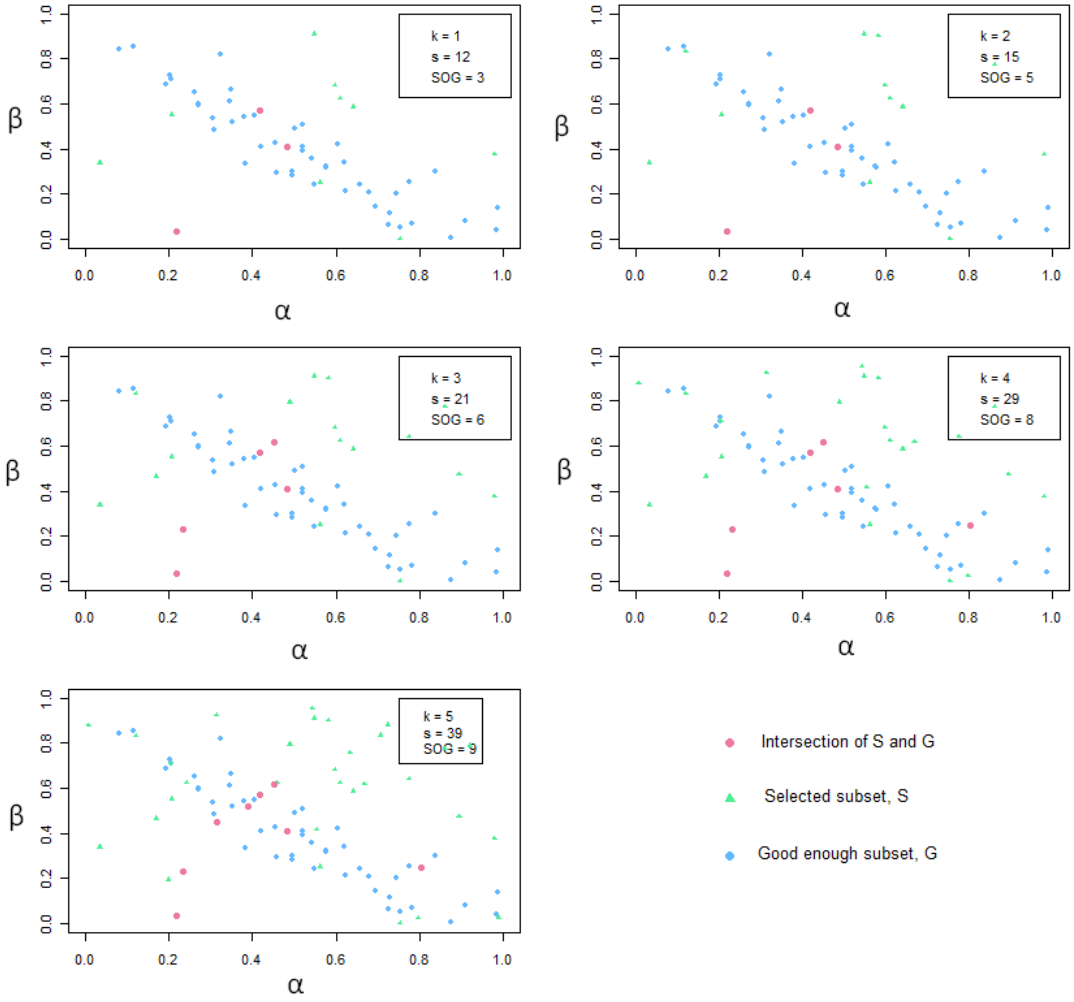

Figure 3.31: α and β values of the good enough subset and the selected subset for the various values of k when a storage capacity is included for ordinal optimisation

between designs but can, with a certain degree of confidence, state which designs are better than others. The shape of the OPC curve is indicates that there are many intermediate designs which points towards the fact that there are many possible solutions which yield an optimal solution. Due to the efficient narrowing down process promised by OO, one can learn from a one-step application of OO the properties of good solutions, such as which portion of the design space is to be explored next, or what a better representation of the problem should be [65]. When the OO procedure was implemented to the simulation model with a storage capacity included the selected subset was very varied. It is noted that although this method has worked well for this test case simulation model, it is mainly aimed at an optimisation technique for complex simulation models. It is, however, noted that the results generated from this method are not as good as those generated from the the other heuristic methods, namely the genetic algorithm.

3.4 Conclusion

It is important to note that this simulation optimisation model represents a simplification of the OCGT plant model and in no way can be used to make real-life decisions. Also, all the units that were used in this model were arbitrary. The following conclusions can be made about the simulation model and the five simulation optimisation methods that were used.

- As expected, the longer the simulation optimisation models were run for, the better the optimum results achieved.
- All methods achieved convergence in parameter values as well as average cost value whenever the model is run.
- For all the methods it was seen that, although the optimum average costs were similar, there seemed to be many parameter values that could achieve this result. In other words, there seemed to be many physically near optimum solutions and the heuristic solution methods were very good at picking this up which is an advantage of these methods.
- When looking just at the three heuristic methods that were applied, the genetic algorithm produced the best results followed by simulated annealing and then the nested partitions algorithm. Best results were defined as the lowest average cost value. Figure 3.32 shows the possible combinations of the 10 runs for each of the models without a storage capacity. It appears in this graph that although there are many possible parameter value combinations, they do form an elliptical pattern in the way in which they optimise. This is consistent with what was expected when the output graph was plotted (see Figure 3.5) and i picked up well by the heuristic methods. The genetic algorithm parameters for the 10 runs appear to be more concentrated than the parameter values for the other two methods.
- In terms of running time, the algorithm that ran the fastest was the nested partitions method with an average of 6 seconds. This was followed by simulated annealing in an average of 185 seconds and then the genetic algorithm in an average of 975 seconds. The NM procedure requires decisions to be made at various stages of the procedure and therefore it is difficult to calculate the exact running time of the process. The same can be said for the OO procedure.
- The nested partitions method is good at giving initial estimates of the optimum results but it was found that a better optimum was found with the other methods but at the sacrifice of an increased computational time.
- For the example of where there was an increase in storage capacity, it did cause a decrease in the optimum average cost value. This was seen to be due to the fact that if there was an oversupply, the excess could be kept in storage and should there be an under supply, the stock in storage could be used. It therefore seems advisable to have a storage capacity rather than not having one. All the methods handled the increase in capacity quite well and therefore showed that these methods were applicable to another simulation model.

- Of all the methods that were implemented, the best solutions were achieved by the genetic algorithm, followed by the simulated annealing algorithm. The nested partitions method performed well in a short amount of time. The NM method did not produce optimum answers as well as the ordinal optimisation method. Table 3.23 shows the optimum results achieved by each of these methods as well as the time it takes to run each of these methods (where applicable).

Method	α	β	Average cost (one run of simulation model)	Time to run
Genetic Algorithm	0.585	0.270	1.448	975 seconds
Simulated Annealing	0.383	0.456	1.549	185 seconds
Nested Partitions	0.804	0.059	1.905	6 seconds
OO procedure	0.726	0.063	1.728	NA
NM procedure	0.85	0.07	2.007	NA

Table 3.23: Comparison of results from different methods

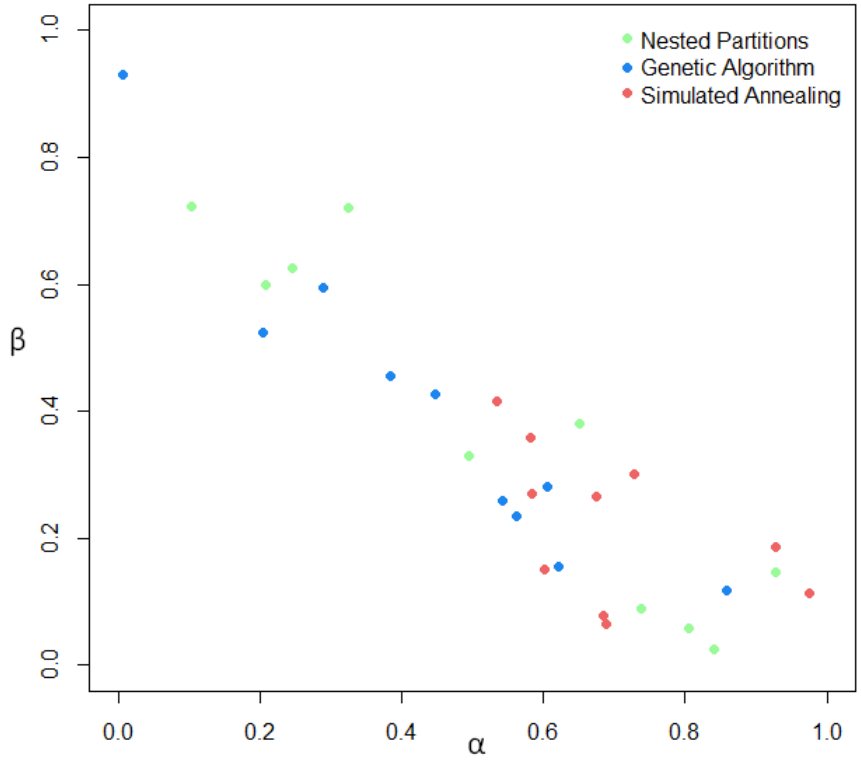

Figure 3.32: Parameter Value Combinations

Table 3.24 shows the advantages and disadvantages that have been seen for each of the 5 methods that have been seen when implementing these optimisation techniques to the simulation model.

Method	Advantages	Disadvantages
Genetic Algorithm	Can search the whole feasible region Does not require decisions to be made while running Provides consistent results	Requires tweaking prior to implementation Takes a long time to run
Simulated Annealing	Can search the whole feasible region Does not require decisions to be made while running	Requires tweaking prior to implementation
Nested Partitions	Runs very quickly Does not require tweaking prior to implementation Can search the whole feasible region Does not require decisions to be made while running	Becomes exponentially difficult to implement when there are many parameters at hand
Nelson-Matejcik	Gives a group of feasible solutions Does not require tweaking prior to implementation	Requires decisions to be made while running
Ordinal Optimisation	Gives a group of feasible solutions Does not require tweaking prior to implementation	Requires model validation Requires decisions to be made while running

Table 3.24: Advantages and disadvantage from the various simulation optimisation methods

Further research is now done by applying these optimisation techniques on a more complicated simulation model.

Chapter 4. Simulation Optimisation of Real Life Simulation Model

4.1 Application to Resource Allocation Problem - Real World Stochastic Simulation Model

4.1.1 Explanation of Model

In the previous chapter, five simulation optimisation techniques were explored and tested on a test case simulation model. Now that these techniques have been understood, they are applied to a real world stochastic simulation model. The real world stochastic simulation model has been designed by an outside client in the energy industry and models the energy production of a total of 14 of the energy producing plants that the client owns. The system that the simulation model models is a resource allocation/stock order problem whereby enough stock needs to be delivered so that it can be turned into an energy supply in order to meet the demand requirements. As such, the main objective is to provide instruments of statistical and simulation nature to minimise the risks in ensuring the provision and delivery of stock with effective working capital management. Although the client had a simulation model which captured the essence of the system at hand, they came to us with the hope of being able to optimise this real world stochastic simulation model by selecting the correct variable values for the simulation model to achieve a desired goal.

Overview and Aim To ensure the adequate delivery of energy supply from the plants that the client owns, a back up of stock is required at the plants. This total stock is made up of the stock on hand as well as any replenishments received. As a result, the total stock level reflects the variation in the stock level based on the amount of stock used and the replenishment of the stock. The total stock was defined to fulfill most of the energy requirements for the plants, with a hedging position at acquiring the shortfalls from the short to medium term markets. However, with the growth in demand, pressure has been put on these markets to deliver the ever increasing shortfalls. The lack of additional capacity has also been a contributing factor. Therefore, the level of stock is at risk and careful management of these has become very much important. It was for this reason that the simulation model was constructed so that the functioning of the plants owned by the client could be modelled and analysis of these stock levels could be conducted.

The simulation model provides 'what-if' scenarios and is forward looking. This means by running the simulation model using defined input values, the output values can be observed and an understanding can be generated in terms of what happens when the input values are defined in a certain way. This model is based on stock supplied, the maintenance plans for each plant, the quantity of stock and the dispatch plan in terms of cost. This means that through controlling the values of the inputs, the simulation model can be run and an output is generated which models how the plants owned by the client would perform in real life provided that the input values were as they were defined to run the simulation model. An example of a what if situation would be to set the simulation input values to reflect no maintenance plan taking place and very little stock of a very low quality being delivered and running the simulation model with these input values. The output from this simulation run would then be able to show what would happen if the input values were defined in such a way.

In order for the simulation model to be constructed to provide realistic future scenarios, an understanding of the cause and effects of the way in which the plants functioned was needed. To do this, the client created and developed a large cause and effect diagram and collected data to create the underlying statistical models used.

The real world stochastic simulation model is a dynamic model based on Monte Carlo simulations. During the simulation, a defined number of iterations of possible demand levels are generated, based on the specified probabilistic model. The aim is to ensure that the various plants that the client owns produce energy so that supply could match demand as much as possible using stock on hand while at the same time minimising the cost of producing this energy as much as possible and minimising the amount of excess stock on hand. This is done by correctly selecting input values for each of the client's plants.

The inputs and outputs used and generated by the simulation model are now explained, once these are understood, the goal that is to be optimised is defined and an analysis of possible alternatives is performed to gain knowledge of the simulation model, after which the simulation optimisation techniques are applied to the simulation model.

Inputs and Outputs In order for the simulation model to run, various input values need to be defined. The simulation model then iterates a predefined number of times using these input values and output values are then given. The horizon over which the inputs and outputs are defined is for the short term and as such the horizon is set to be equal to a period of one year. This yearly period is broken down into months and as a result the input values for each plant are defined for each month. Some of these values change from month to month and others are constant for each month. Similarly, the output values given are defined for a period of one year which is also broken down into monthly periods. Due to the outputs being dependent on the inputs, they are effected by the changes in some of the monthly input values and as such are always defined on a monthly basis. An example of an input that varies on a monthly basis is stock delivered to a plant while an input that varies on a yearly or longer period is the merit order of the cost allocated to a plant. The merit order defined as a way of ranking available sources of energy in ascending order of their short-run marginal costs of production so that

those with the lowest marginal costs are the first ones to be brought online to meet demand and the plants with the highest marginal costs are the last to be brought on line. In terms of outputs, one that varies monthly is the excess stock at the end of each month.

For confidentiality purposes, a full list of the inputs and outputs cannot be given. However, Table 4.1 shows the number of inputs and outputs respectively that change either on a monthly period or a yearly or longer period. These inputs and outputs are defined for each plant for each month (if a input or output only changes over a yearly period or longer the monthly value remains constant throughout that period).

	Number of inputs	Number of outputs
Monthly change	10	23
Yearly of longer change	10	0

Table 4.1: Relevant input and output values

The inputs values are defined in the production plan. This production plan is then used by the simulation model which then runs and generates the output values.

4.1.2 Hardware and Software Used

The hardware that was used to run all the simulation optimisation models is a Intel(R) Core(TM) i5 CPU processor with and 8GB installed memory (RAM). Various software was used through the running of the simulation optimisation model.

The simulation model itself was designed and provided by the client. It was created and run using Dynamic Information Architecture System (DIAS). DIAS is a flexible, extensible, object-based framework for developing and maintaining complex multidisciplinary simulations. The DIAS infrastructure makes it feasible to build and manipulate complex simulation scenarios in which many thousands of objects can interact via dozens to hundreds of concurrent dynamic processes. The model reads in an excel document where each sheet corresponds to a plant and contains the relevant input values for that plant. The simulation model then runs with these inputs and when the simulation horizon has been reached an excel file containing the output values for each plant on a separate sheet is produced.

The optimisation techniques were manually coded in R version 2.15.

In order to be able to run the simulation optimisation model, a link between the simulation model and the optimisation model needed to be created. This is done using the R package called XLConnect (version 0.2-1). The purpose of this package is to provide a way to manipulate Excel files directly from the R interface. The simulation model itself was invoked from R using the system() command which invokes the OS command prompt.

Figure 4.1 shows the simulation optimisation framework. The software, written in brackets, is shown at the various stages at which it is used. In terms of running one cycle shown in Figure 4.1, most of the computational time is spent running the simulation model followed closely through the XLConnect link between R and DIAS.

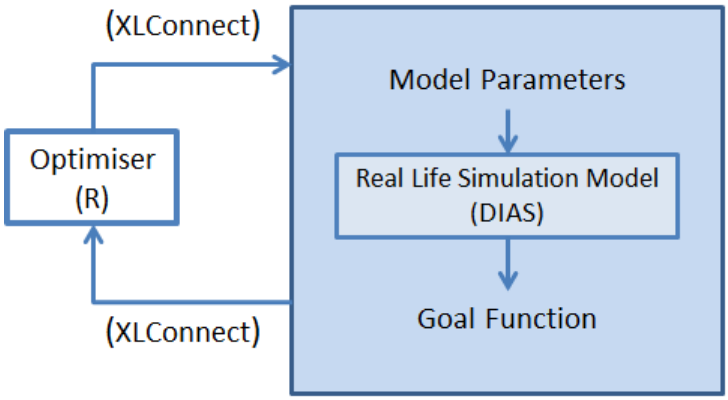

Figure 4.1: Simulation Optimisation Model with software

It is noted that the time it takes for one run of the simulation model to complete is 46 seconds.

4.1.3 Variables

At this point, the simulation model is developed and fully operational but has no optimisation capability. There are many input variables in the simulation model. In order for optimisation techniques to be implemented, a decision about which of these inputs could be defined as variables needed to be made.

To decide this, a workshop was held with the client. The aim of the workshop was therefore defined as to seek guidance as to how the simulation model should be used to address the questions of 'optimal' stockpile levels while at the same time giving consideration to what the controllable policy actions are as well as what the key management objectives are. To achieve this, the question of 'what are the relevant policy variables?' was asked. It was important to establish this so that an understanding was gained in terms of the inputs that could be changed in order to reach the optimum value. This was because the simulation model takes in many input values but it is important to know which of these values are fixed and which can be changed. So in other words, which of the input values are controllable by the client and which are fixed and not controllable.

The workshop was held at the client's offices over the course of one day. The workshop started out by giving a brief overview of the simulation model, the inputs and outputs that it has, as well as its internal structure. This was to ensure that everyone was well informed as to the functioning of the simulation model. This followed by an explanation as to the key concepts underlying the search for 'optimal' policies using simulation models.

Many stakeholders were present from the various departments of the client. This was done to ensure that when the brainstorming session took place, the various objectives of the different stakeholders were all kept in mind. Also, having a representation of the various stakeholders meant that different points of view could taken into consideration.

From this a brainstorming session started to identify the answer to the following two questions; what are the key policy objectives to be optimised and what are the relevant policy variables? The reason that the second question was asked was because although there might be many key policy objectives, some might not yet be captured by the simulation model and some may be a long-term objective and therefore not a priority at this moment and it is important to capture this. The way in which this brainstorming took place was to take one question at a time and everyone in the room was given the opportunity to answer the question independently. The room was then circled, giving everyone the opportunity to provide one of their answers at a time. This was done until everyone had an opportunity to voice all their answers. The reason as to why everyone could only give one answer at a time was to ensure that most people got an opportunity to respond and also to ensure that the process was moving and engaging.

This method is known in the literature as the Nominal Group Technique. This is a decision making method for use amoung groups of many sizes, who want to make their decisions quickly while at the same time take everyone's opinion into account. This method was selected due to its many advantages. Some of these are that it is easy to learn, applicable to a wide variety of areas and situations, easy to integrate into programs and projects of a larger scope, highly satisfying to participants and quite successful at inspiring a commitment to action, follow through and follow up [99].

Lastly, the ideas that were established were then clustered and structured into policy actions and objectives as well as across the short term (less than one year) and longer term (more that one year) horizon. This information was then used when trying to define the objective that the simulation optimisation model must minimise.

From this workshop, the client was able to identify long term policy modifications as well as short term policy modifications and potential later modifications. Since the horizon which is to be looked at during the simulation optimisation is of a period of one year, and therefore short term, only the variables which are defined over the short term are considered. The short term policy modifications were defined as moving maintenance, merit order of the cost of each plant, quality deviations in stock and utilising contract variabilities when dealing with stock delivery. The variables associated with these policy modifications are explained below. At this stage, it is worth recalling that the client owns 14 plants, each with the same number of variables.

1. Planned maintenance

    - The input value is defined as the planned maintenance.
    - This input is defined as the percentage of time for the year that the plant can be closed for planned maintenance. It is noted that when the plant is closed for maintenance, no supply can be generated.
    - For the purposes of this simulation optimisation model it has been decided that the yearly total for the plant maintenance of each plant is defined before the model is run but the way in which this is distributed over each month for each plant can vary.

- There are 14×12 variables associated with planned maintenance. (14 plants with a value for each month of the year.)
- For the year under review, it has been decided through the production plan that the average planned maintenance over all 14 plants is 10.45%. Table 4.2 shows the yearly percentage that each plant needs to be closed for planned maintenance.
- There is a limit, however, as to how much planned maintenance can happen at a plant each month and that is 50% of the time that the plant is run for each month. This is because each plant is required to at least be actively working 50% of each month. This constraint is never imposed with the production plan at hand as none of the plants have a yearly planned maintenance of 50% and therefore this would never result in a plant needing to be closed more than 50% of the month.

Plant	Yearly planned maintenance (%)
A	4.83
B	16.78
C	7.35
D	5.84
E	12.89
F	15.51
G	9.88
H	8.25
I	6.32
J	8.13
K	10.84
L	7.92
M	17.98
N	13.66

Table 4.2: Yearly planned maintenance values for each plant

- An assumption at this point is made. Due to the fact that there are a different number of days in each month, x% of maintenance occuring in a month that has 31 days is more than x% of maintenance occuring in a month that has 30 days. At this point however, it is assumed that this difference is negligible.

2. Merit order of unit

- The corresponding input is the 'cost' of supply.
- The reason that 'cost' is in inverted commas is because the cost of supply is not defined in monetary terms but rather as a merit order. What this means is that the 14 plants get ranked from 1 to 13 where 1 is allocated to the plant that is the cheapest to run and 13 is given to the plant that is the most expensive to run.
- The reason that there are 13 ranking orders and not 14 is because although there are 14 plants, 2 are on the same site (essentially, they are one plant split into two) and as a result, they have the same cost of supply and therefore are given the same ranking order. These are plants J and K.

- The ranking of the costs is fixed throughout the year period for which the simulation model runs.
- There are a total of 13 variables associated with 'cost' of supply.
- The cost of each plant can be controlled through parameters that the client can control but are beyond the scope of this project.

3. Utilise contract variabilities

- The corresponding input is the nett stock delivered to each plant each month.
- The minimum amount of stock that a plant can receive a month is 0 and the maximum amount is specific to each plant and depends on the storage capacity of that particular plant.
- There are a total of 14×12 variables associated with the stock delivery. (14 plants with a value for each month of the year.)
- The monthly stock delivery for each plant is constrained to have a minimum of 0 and a maximum equal to the monthly plant capacity. This creates a very large feasible region for the amount of stock that can be delivered to the plant each time. These constraints are shown in Table 4.3. Although the range for each parameter is quite large, it was desired to not tighten the constraints so that the whole range could be taken into account when performing that analysis. Further research from the client's side might be to look into how these constraints can be tightened which would then lead to a smaller feasible region to be explored.
- It is also noted that the simulation model has an inbuilt constraint whereby no more than 3000 units of stock can ever be delivered to a plant in a one month period.

Plant	Minimum monthly delivery (units)	Plant capacity/Maximum monthly delivery (units)
A	0	2232
B	0	1440
C	0	3450
D	0	1110
E	0	1865
F	0	3558
G	0	3690
H	0	3840
I	0	853
J	0	1425
K	0	1425
L	0	3843
M	0	3450
N	0	3510

Table 4.3: Nett stock delivery constraints for each plant

4. Quality deviations in the stock fed to plant

- The corresponding input is the quality of the stock.
- This value is bounded based on what the quality value the stock can take on. From research it is seen that the quality of stock can range between 17 and 22 units.
- Although this value can change from month to month, the normal operations are that these values are set by management for the year. As a result, these values are selected to remain constant for each plant throughout the year. Further research might lead to these values changing monthly.
- There are 14 variables associated with the quality value.

To summarise, the model has a total of 700 variables which are broken down as:

- 1 yearly quality value for each plant totaling to 14 quality values.
- 1 yearly cost value for each plant totaling to 14 values.
- 1 monthly planned maintenance value for each plant totaling to 12 a year for 14 plants giving 168 values.
- 1 monthly stock delivery value for each plant totaling to 12 a year for 14 plants giving 168 values.

4.1.4 Objectives and Goal

The aim of running an optimisation model is to achieve the 'best' outcome by finding the particular alternative that leads to this 'best' outcome. In order to measure the strength of the outcome produced, a goal value needs to be calculated based on the outputs generated from the simulation run. The aim of this goal value is to capture the aspects of the alternative that are significant when trying to find the best alternative. This goal value is then used to compare one alternative to another in order to find the alternative that 'best' achieves the outcome desired. Before a goal can be defined, the objective or objectives that need to be captured within the goal need to be defined. If there is only one objective, such as in the test case simulation optimisation model, then this objective itself is the goal. If, however, there is more than one objective then the goal becomes a multi-objective one and is designed in such a way that it captures all of the objectives, such as in the real world simulation optimisation problem. For this particular simulation model, there are three objectives that are focused on and that are used to build up the goal value. These are listed below followed by a justification as to why these quantities were included and how they are calculated. It is noted that these three values are defined for a period of one year across all 14 plants.

- Cost of production
  - Cost of production is to be minimised in the sense that whenever supply is generated it is done so in the cheapest way possible.

- Cost of production is defined using a merit order, which, as previously defined, is a way of ranking available sources of energy in ascending order of their short-run marginal costs of production, so that those with the lowest marginal costs are the first ones to be brought online to meet demand, and the plants with the highest marginal costs are the last to be brought online.
- From the section on variables, it is noted that each plant is ranked according to their costs where 1 is assigned to the plant that is the cheapest to run and 13 is assigned to the plant that is the most expensive to run. This means that the cost of producing 1 unit of supply at the most expensive plant is 13 times higher than producing 1 unit of supply from the cheapest plant. It is noted at this point that this is a simplifying assumption. The cost of production is calculated for each plant by multiplying this cost value by the simulated supply value produced. This gives a cost of production for each plant for a period of one year. These costs are then summed across all plants giving a cost of production for all plants for a period of one year.

$$\text{Cost of production} = \sum_{i=1}^{14} y_i r_i$$

  where y_i is the amount of supply produced, r_i is the cost rank allocated to the plant i.
- It is agruable as to whether or not this is the best way to represent the cost of production. This is due to the fact that ordinal raking values do not have any cardinal meaning. However, this is the way that the client classified their cost of production and as a result was carried through when performing this analysis. It is noted that the main concern in this thesis is the quality of the optimiser applied to the simulation model and as such this classification of cost of production is not of importance at this stage. Further research into this classification scheme could be done at a later stage.

- Excess stock on hand
  - Excess stock on hand is chosen to be included in the goal as it is extremely costly to keep stock on hand if it is not required. The aim is then to minimise having large stockpiles at each plant. Ideally, what is wanted is there to be enough stock to produce supply to meet demand and perhaps with a little bit more to act as a buffer if actual demand is far off from expected demand.
  - The excess stock is defined for each plant as the sum of the stockpile days for each month over the period of one year. To get the total excess stock, the excess stock for each plant is summed. This means that the excess stock value is the total number of stockpile days that were had by all the plants at the end of the month over the

period of one year.

$$\text{Excess stock on hand} = \sum_{i=1}^{14}\sum_{j=1}^{12} x_{ij}$$

where x_{ij} is the excess stock at hand for plant i in month j.

- Unmet demand of energy
  - One of the inputs that go into the simulation model is the planned demand for each plant each month. These values are defined by the client and make up one of the inputs that go into the simulation model. One of the objectives is to then produce enough supply to meet the actual demand which fluctuates around the planned demand with some probability.
  - The unmet demand of energy for each plant is then calculated as the planned energy for the year less the simulated energy for the year for each plant. The total unmet demand is then defined as the sum of the unmet demand of energy over all the plants.

$$\text{Unmet demand of energy} = \sum_{i=1}^{14}(\text{Planned energy}_i - \text{Simulated energy}_i)$$

where i is the plant number.

Since there are three objectives to consider when achieving the goal, a consistent and accurate approach needs to be taken in order to combine them together to create the multi-objective goal. The reason behind combining all these objectives is to make sure that they are all taken into account when performing the optimisation. If focus was only put on one objective at a time, achieving that particular objective might lead to the other objective completely not being met, particularly if some objectives are contradictory. Therefore, combining all objectives into a multiobjective goal ensures that each objective is satisfied as best as possible. In order to do this it is important that each of the three quantities are commeasurable. This is done by creating a 'payoff' table [34] as shown in table 4.4. This table is generated by optimising (in this case minimising) each objective in turn to get the ideal value for each objective. This ideal value is noted as well as the value that the other objectives take on.

To construct this payoff table, the simulated annealing algorithm was used to optimise each objective. Simulated annealing was chosen as the optimising technique as out of all four optimisation techniques available, it is the technique that is the fastest to run. It is worth noting that certain of the ideal values are not what one would expect. For example, if looking at cost of production in isolation, one would expect it to have a minimum value of 0 when no supply is generated but due to the constraints in place this is not possible as the simulation model is designed to produce supply to meet actual demand which therefore leads to a cost being incurred. It is also noted that to achieve the ideal value of one objective, the ideal value as well as the other objective values can fluctuate due to the fact that there is noise in the simulation model. For example, it is possible to have a minimum overstock and a very large

Objective being minimised	Values obtained for:		
	Cost of supply	Yearly overstock	Undersupply
Cost of supply	**1148669**	3002	14317
Yearly overstock	1382331	**0**	19870
Undersupply	1354283	2800	**1374**
Range of values	233662	3002	18523

Table 4.4: Payoff table for real world stochastic simulation model

cost of production value and it is also possible to have a minimum overstock and a very small cost of production, both of these two outcomes can have a very significant impact on the range of each of the objectives.

At this point, an assumption is made that the ideal values, as well as the ranges for each of the objectives shown in Table 4.4, are adequate for the purposes of the goal construction. For the purposes of this thesis these are seen as good enough solutions.

Now that the range of values that the three objectives can take after minimising each one in turn has been defined, the scales of each of the values can be adjusted to some comparative scale. This is done by dividing each objective by the range that that objective can take on. By weighting all the objectives, all performance measures are scaled to have a range of 1 unit from best to worst in the payoff table and places equal importance on achieving going from the worst to the best outcome for each of the objectives.

These weighted objectives were then combined to get one value. It is noted that some of the above listed objectives are more significant than others. As a result a weighted sum of the objectives needs to be taken which ensures that all objectives are commeasurable and that the desired relative importance of the objectives is captured. The objectives that are more important will get larger weights than the objectives that are less important. For this particular simulation model, the objective of most importance is ensuring that there is as little unmet demand as possible. As a result this is given the largest weight. The next important objective is to minimise the cost of production followed by having a small excess of stock. The weight breakdown was then decided as follows: unmet demand is weighted 50%, cost of production is weighted 30% and excess stock is weighted 20%. The weighted sum is then defined as:

$$\text{weighted sum } = \sum_{i=1}^{3} w_i z_i$$

where z_i is the value of objective i and w_i is the weight associated with objective i. It is noted that w_i is a fraction value where the numerator corresponds to the importance weight of the objective and the denominator corresponds to the weight that is used to scale the objectives so that they are commeasurable. As a result the effective weights for each unscaled objective are defined as $\frac{50}{18523}$ for undersupply, $\frac{30}{233662}$ for cost of supply and $\frac{20}{3002}$ for yearly overstock.

By multiplying each objective by these effective weights it creates objectives that are not only scaled but also commeasurable.

Now that the above weighted sum has been defined as a sum of the appropriately scaled objective values, the final goal value can be specified. It is noted that, most of the time when performing optimisation, the weighted sum itself would be taken as used as the goal value to be minimised. However, it has been noted by Miettinen in [73] that using this weighting method does not work for non-convex problems. She goes on to say that it is important in multiobjective optimisation that Pareto optimal solutions are generated and that any Pareto optimal solution can be found. In this respect, the weighting method has a serious shortcoming. It can be proven that any Pareto optimal solution can be found by altering the weights only if the problem is convex. Thus, it may happen that some Pareto optimal solutions of nonconvex problems cannot be found no matter how the weights are selected. This is a serious and important aspect because it is not always easy to check the convexity in real applications if the problem is based, for example, on some simulation model or solving some systems like systems of partial differential equations. If the method is used in nonconvex problems for generating a representation of the Pareto optimal set, the goal gives a misleading impression about the feasible solutions available when some parts of the Pareto optimal set remain uncovered.

To overcome this challenge of using a weighted sum to calculate the goal, the method of global criterion is used. This method minimizes the distance between some desirable reference point in the objective space and the feasible objective region. For the purposes of this case, the reference point used is the ideal value that was calculated and shown in the payoff table for each objective. The method of global criterion can be defined as

$$\text{minimise} \max_{i=1,2,3} \frac{(z_i - I_i)}{q_i}$$

where z_i is the value of objective i, I_i is the ideal value of objective i and q_i is the range of possible values for that objective obtained from the payoff table. It is important to note that since the three objectives have different magnitudes, the objective functions are scaled by dividing each value by the range for that objective identified in the payoff table.

In order to ensure that the relative importance weights are captured, the weighted sum multiplied by some small ϵ (defined to be 0.1 in this case) and added to the value defined by the method of global criterion. Therefore, the final goal value is defined to be:

$$\text{minimise} \left\{ \max_{i=1,2,3} \frac{(z_i - I_i)}{q_i} + \epsilon \sum_{i=1}^{3} w_i z_i \right\}$$

where ϵ is equal to 0.1, I_i is the ideal value of objective i, z_i is the value of objective i, q_i is the range of possible values for that objective obtained from the payoff table and w_i is the weight associated with objective i.

This goal value is to be used in the remainder of this thesis as the value which is to be minimised when applying the optimisation technique to the simulation model. In other words,

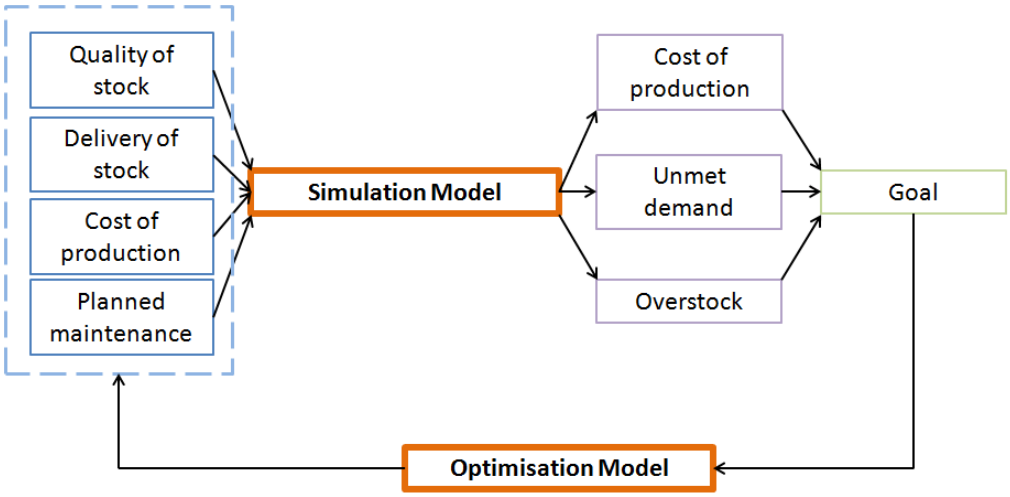

Figure 4.2: Overview of the simulation optimisation process

the aim of the optimisation method is to minimise this value which is generated when running the simulation model for a defined period of time. This goal value is used as it captures all the three identified objectives correctly while providing a single value.

Lastly, it is noted that the actual goal value cannot be interpreted to mean anything, it is just a numerical combination of the categories that are important and need to be minimised. In order to get a firm understanding of the goal, it needs to get unpacked into the three objectives of cost of production, excess stock on hand and unmet energy demanded.

4.2 Optimisation Problem

Now that the various alternatives have been implemented, which has allowed better understanding of the simulation model, and the goal value that is to be optimised has been decided, the optimisation techniques can be implemented. Figure 4.2 shows the process that is undertaken through the simulation optimisation. This shows the inputs taken in, the simulation model run, the objective values calculated, the goal value generated and the optimisation model running.

From the previous chapter, 5 simulation optimisation techniques were explored. These were Simulated Annealing, Genetic Algorithms, Nested Partitions, Ordinal Optimisation and lastly, the Nelson-Matejcik optimisation method. All of these methods, with the exception of the Nested Partition method, were implemented and explained in detail below. The same terminology defined in the previous chapter, as well as the way in which these methods were implemented, will be used when applying these methods to the real world stochastic simulation model.

4.2.1 Nested Partitions

The Nested Partitions method worked very well for the test case simulation model as it gave fast and accurate results which would indicate that it should be used on the real world stochastic

simulation model. However, due to the number of variables that the real world stochastic simulation model has, this is not feasible as the problem grows exponentially. This can be explained as follows (using the same terminology as the previous chapter); suppose that for each variable, the range of the most promising region is always divided by 2 giving 2 partitions and the range of that variable not in the feasible region makes its own partition, M. This means that each variable's range is divided into 3 partitions.

If there were only two variables being considered, as with the test case simulation model, then there would be $2^2+1 = 5$ regions to consider at time. This is made up of the possible combination of the promising region for each variable and then the extra region which corresponds to the remainder of the feasible space to prevent the problem from getting stuck at a local minimum. This would lead to a randomly selected combination of variables for each region being chosen and the simulation model run with these values and would result in a total of 5 goal values and the process would be repeated. Suppose there were now 3 variables instead of 2. Then the total number of regions to consider at a time would be $2^3 + 1 = 9$. This formula can then be generalised such that if there were n variables to consider there would be a total of $2^n + 1$ regions to consider, from each of which a solution would be generated.

For the real world stochastic simulation model, there are a total of 700 variables. This would mean that there were a total of $5.2601e{+}210$ regions to consider which exponentially is too large to be carried out on the computer software available.

It is noted that parallelisation of this method is a possibility. This would mean breaking up the large problem into smaller ones which can be solved concurrently. Due to the number of variables in this simulation model, however, the simulation model would need to be broken down into many smaller problems in order for the nested partitions algorithm to be applicable and not grow exponentially large. The risk then runs of having so many smaller problems that the purpose of the nested partitions algorithm is defeated.

As a result, although the Nested Partitions method works well when there are a small number of variables to consider at a time, the method is not adapted for use when there are a large number of variables in the simulation model being used and therefore this method is not implemented. Should the number of variables decrease in the simulation model (perhaps if focus on only a particular aspect of the model is needed) then implementing the nested partitions algorithm in parellel would be a further point to research.

4.2.2 Simulated Annealing

The next simulation optimisation technique that was to be implemented on the real world stochastic simulation model was that of simulated annealing. The outline of this simulation optimisation method is given in Figure 3.10 and is based on the thermodynamic cooling of molten metals. This simulation optimisation technique is used with the aim of minimising the goal defined above by choosing the correct variable values.

Preliminary Information

Before the simulated annealing technique is implemented, the following model specifications need to defined:

- The neighbourhood structure
- The acceptance probability
- The initial temperature
- The cooling schedule
- The stopping rule

Each of these are now explained in detail below.

The neighbourhood structure The neighbourhood structure is important as it defines the area around the current solution from which the second solution will be generated. This needs to be selected carefully as a small neighbourhood allows for fast convergence but if the neighbourhood is too small then it might lead to convergence to a local optimum. For this real world test case, the neighbourhood structure is always selected to be symmetric and also defined in such a way that it never allows parameters to be defined outside of their defined range. Due to the fact that there are continuous parameter values, discrete parameter values and continuous parameter values that need to sum to a correct value, the neighbourhood structure for each of these parameters are defined differently.

- *Continuous parameters* (Stock delivery and quality) - The neighbourhood structure is set to $\pm 10\%$ of the allowable range from the current parameter value.
- *Discrete parameters* (Cost of production) - The cost of production variable is a discrete ranking order variable ranged from 1 for lowest cost to 13 for highest cost where 2 plants are allocated the same cost value. The neighbourhood structure for this variable is defined by randomly selecting three plants, getting the cost value associated with each of these plants, randomly changing the order of these values and then returning this new combination to the three plants that were randomly selected. This means that the second solution will still have the same ranking for all but three of the plants as the current solution. This is illustrated in Figure 4.3. The figure shows the cost ranking for each of the 14 plants for the current solution and the second solution after plants C, E and I were randomly selected. It is noted that the 2 plants that have the same cost of production are treated as one in this case to ensure that they adhere to the requirement that they are both ranked with the same cost.
- *Continuous parameter values that need to sum to a correct value* (Planned maintenance value) - A similar approach was taken for this variable as for the cost of production variable. For the planned maintenance values for each plant, the monthly values can

	Plant													
	A	B	C	D	E	F	G	H	I	J	K	L	M	N
Current solution cost allocation	1	9	**8**	5	**11**	3	2	12	**13**	10	10	7	4	6
Neighbourhood solution cost allocation	1	9	**13**	5	**8**	3	2	12	**11**	10	10	7	4	6

Figure 4.3: Example of the the neighbourhood structure associated with the cost of production parameter for all plants

change but the monthly values summed to give the yearly total needs to be equal to a predefined value. When defining the neighbourhood structure, 3 months were randomly selected, their planned maintenance values were summed together and then randomly split across the three plants. This means that the monthly values for three of the months were different for the current solution and the second solution. An example of this process is illustrated in Figure 4.4. For this particular plant, February, September and October were randomly selected, their values summed and then spread over the three months again randomly. It can be seen that the yearly planned maintenance value for both the plants is equal to the required amount of 16.78, which is the required amount for plant B.

	Plant												Total
	Jan	**Feb**	Mar	Apr	May	Jun	Jul	Aug	**Sep**	**Oct**	Nov	Dec	
Current solution planned maintenance	.65	**.23**	.79	1.12	.95	2.42	.29	2.61	**1.97**	**2.67**	.67	2.41	**16.78**
Neighbourhood solution planned maintenance	.65	**.97**	.79	1.12	.95	2.42	.29	2.61	**2.44**	**1.46**	.67	2.41	**16.78**

Figure 4.4: Example of the the neighbourhood structure associated with the planned maintenance parameter for plant B

The acceptance probability The acceptance probability is important as it is used to decide whether or not the second solution takes the place of the current solution even if it performs worse than the current solution (in the real world stochastic simulation model this would mean that the goal achieved by the second solution is higher than the goal achieved by the current solution). With the same justification as the test case simulation model, the Boltzmann probability function is used. This is defined as

$$P(f(s), f(s^{'}), T) = \begin{cases} 1 & \text{if } f(s^{'}) - f(s) < 0 \\ exp(-\frac{f(s^{'}) - f(s)}{T \times K_B}) & \text{if } f(s^{'}) - f(s) \geq 0 \end{cases}$$

Where $f(s)$ is the current solution, $f(s^{'})$ is the second solution, T is the temperature and K_B is the Boltzmann constant (defined as 1.381)

The initial temperature As the simulated annealing algorithm is run, the temperature of the system is decreased which in turn decreases the probability with which a second solution that has a higher goal value than the current solution is accepted. It is therefore important that a good initial temperature value is selected so that at the beginning of the algorithm virtually all moves are accepted and as such the Boltzmann probability needs to be equal to 1 at the starting temperature which allows for maximum exploration of the neighbourhood.

Therefore, using the definition of the Boltzmann probability above, in order to calculate the temperature at which the probability of accepting a solution is 1 when $f(s^{'}) - f(s) \geq 0$ we need to calculate $f(s^{'}) - f(s)$.

If running time was not a constraint, the simulation optimisation model would be run many times with different parameter values so as to get an average of the difference $f(s^{'}) - f(s)$. Due to the fact that running time is limited an approach was taken which, although saves on running time, does mean that the initial temperature value may not be very accurate. Also, since $f(s)$ represents a solution from a simulation model, the noise generated means that the value of $f(s)$ fluctuates.

The value for $\left|f(s^{'}) - f(s)\right|$ was then generated. This was done as follows:

- Populate the 700 variable values for the initial model.
- Run the simulation model using this initial model and calculate the goal of the initial model.
- Populate the 700 variable values for a second model by taking the initial variable values and generating new variable values within the defined neighbourhood space.
- Run the simulation model using the second model and calculate the goal of the second model.
- Calculate the absolute difference between the initial model's goal and the second model's goal, giving $f(s^{'}) - f(s)$.
- Repeat this process 40 times.
- Take the average of these 40 $\left|f(s^{'}) - f(s)\right|$ values and use this median value to calculate the temperature.

The median value of the absolute difference between the initial and second goal values is defined as 0.553.

Now that the value of $f(s^{'}) - f(s)$ is known the initial temperature can be found. Figure 4.5 shows how the Boltzmann probability changes as the temperature value changes. From this it can be seen that the acceptance probability is equal to 1 at around a temperature of 71. This means that the temperature is set to be equal to 72 when the simulation annealing algorithm is run so that the acceptance probability is equal to 1 at the start of the algorithm.

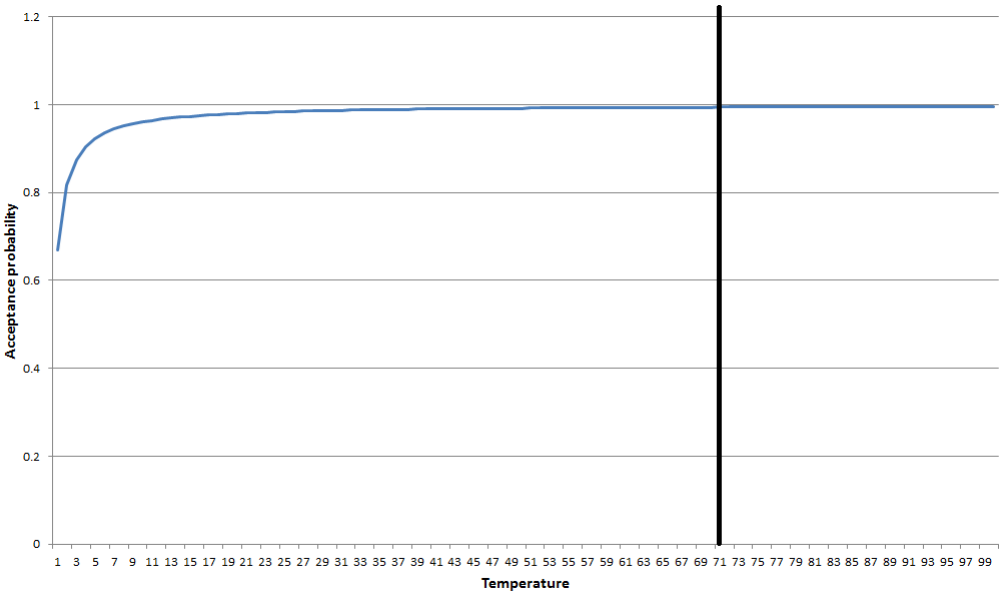

Figure 4.5: Affect on acceptance probability as temperature changes for real world stochastic simulation model

The cooling schedule Similarly to the test case simulation model, a geometric cooling schedule is used which is defined as $T_i = \gamma T_{(i-1)}$ where T_i is the temperature at the ith iteration and γ is the cooling parameter. Figure 4.6 shows the change in temperature from one iteration to the next when a geometric cooling schedule with different cooling parameters are used. It can be seen that the lower the cooling parameter, the faster the temperature reaches 0. Since there is noise present in the solution generated by the simulation model, it is desirable that the temperature does not cool down too quickly as this might lead to rejections of good solutions that just happened to perform badly for that simulation. The cooling parameter that is used when running the simulated annealing algorithm is equal to 0.95. This means that around the 130 iteration of the model, the temperature value is approximately 0.

The stopping rule Similarly to the test case simulation model, the stopping rule that is to be used is that the algorithm is terminated after 1000 iterations have been carried out. Based on the initial temperature, the cooling schedule and the acceptance probability, it is expected that the goal value of the current model will fluctuate and possibly increase at the start of the optimisation process and ideally, by the time the 1000 iteration has taken place, the goal would have converged to the optimum value.

Simulation Optimisation

Now that the preliminary information has been selected, the simulation optimisation can be carried out. The details of this process, similarly to the test case, are now explained.

1. The initial specifications are set.

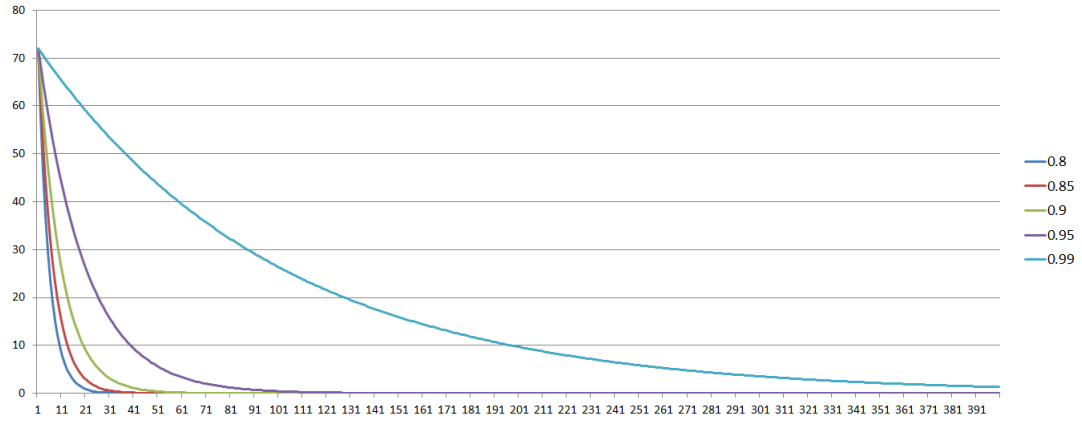

Figure 4.6: Change in temperature from one iteration to the next using a geometric cooling schedule with different cooling parameters

2. The number of times the optimisation algorithm is iterated is defined to be 1000.

3. The initial temperature is set equal to 72.

4. The 700 parameter values are defined each of them within their allowable range (for more information on this see section 4.1.3). It is noted that the initial parameter values are randomly selected and therefore the parameter values can change each time the model is run. Once the 700 parameter values are defined, the model is populated with these values and defined as the initial solution.

5. The simulation model is then run with these parameter values and the goal value is calculated (call this *goal 1*) and the parameter values are saved.

6. The 700 variable values for the second solution model are then calculated. To generate these parameter values, a value is generated from the neighbourhood of the corresponding parameter in the initial solution. The model is populated with these parameter values and defined as model 2.

7. The simulation model is then run using the parameter values defined for the second solution and the goal value is calculated and saved (call this goal value *goal 2*), as well as the parameter values.

8. The simulated annealing algorithm is then run based of the 2 goal values above. Three possibilities can occur:

   (a) If *goal 1* $\leq$ *goal 2* then the current solution is made equal to the second solution; the variable values as well as the goal change accordingly.

   (b) If *goal 1* $>$ *goal 2* then a random number u is generated and two possibilities can occur

i. If $u < \exp\left(\frac{goal\ 1\ -\ goal\ 2}{temperature}\right)$ where *temperature* is defined according to the relevant cooling schedule. Then the current solution then becomes the second solution

ii. Otherwise, the the current solution remains as is

9. The process is then repeated from step 6 until the stopping criteria is met which is that 1000 runs have been completed.

10. At this point, the final parameter values should have converged to a constant value producing the minimum goal value.

After having established the way in which the simulated annealing algorithm was to be run, it was implemented on the real world stochastic simulation model 30 times. The best final solution generated from these 30 runs is shown in Table 4.5. When comparing the objective values to the ideals it can be seen that the results achieved by the simulated annealing algorithm are good. The undersupply and cost objective values are close to their ideal values. Also, it can be seen that the results are much better than those achieved by all six of the alternatives in the alternative analysis.

Goal value	Cost value	Overstock value	Undersupply value
18.278	1152559	2083	5179

Table 4.5: Final solution from simulated annealing - goal and objective values

Figure 4.7 shows the path to convergence that the goal value took. It can be seen that the initial solution had a goal value of about 25. During the first 160 iterations of the simulated annealing algorithm worse moves are accepted in the sense that the goal value increases from one iteration to the next. This is an essential part of the simulated annealing algorithm as it avoids the goal value from converging to a local minimum. After the 160th iteration it is seen that no incorrect moves are accepted. This is due to the fact that the temperature value has reached approximately 0 by then which therefore means that the acceptance probability is 0 and, as such, moves that lead to an increase in the goal value are prevented. This means that any moves that will take place from then onwards will be moves the lead to the goal value decreasing. The simulated annealing algorithm appears to have reached convergence by the 796th iteration, where the goal value is equal to 18.278 since no moves that better the goal value take place from then until the stopping criteria comes into place. It is seen that the range between the highest goal value and the lowest goal value reached by the simulated annealing algorithm is 10.27.

Figure 4.8 shows the convergence patterns of each of the objectives that make up the goal. The objective values shown in the graph have been scaled by their relative weights so that they can be compared. Although all objectives are fairly volatile, the cost value and undersupply values tend to stabilise faster than the overstock value. The volatility in the overstock objective can be explained as follows; out of the 700 parameter values, 168 are used to define the overstock objective. These 168 parameter values correspond to the stock being delivered to the 14 different

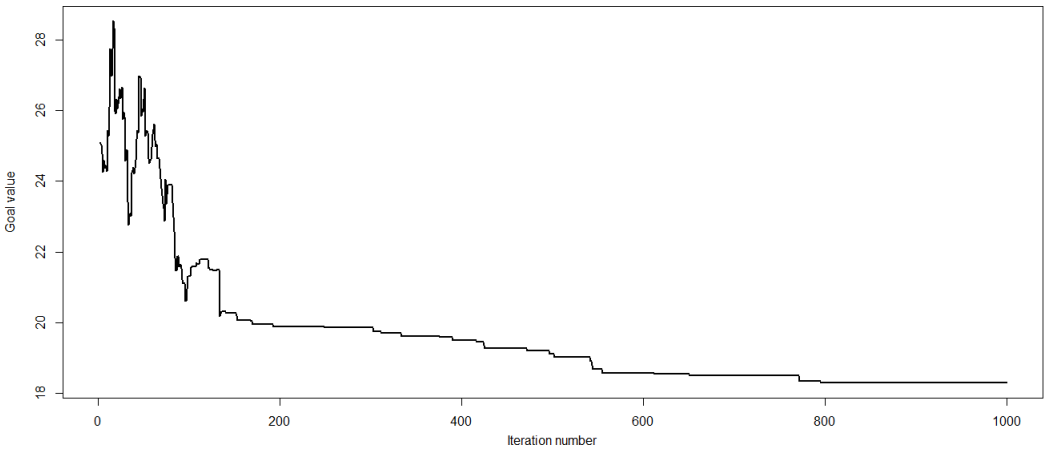

Figure 4.7: Simulated annealing goal value convergence pattern

plants. The parameter values each have a very large range of possible values and therefore the neighbourhood of each of these parameter values is large. This means that from one iteration to the next the parameter values can vary largely and as such there can be a large change in the level of overstock.

The overstock objective has the lowest importance weight in the goal value and therefore it is understandable that changes in this objective value will have less significance on the goal value. This is proven in the fact it can be seen that the large decreases in overstock value that take place between the 400th and 600th iteration do not seem to cause a large decrease in the goal value. It also seems as though the overstock objective and the undersupply objective are more or less inversely related in the sense that when there is a large overstock, there is limitted undersupply. This confirms what would be assumed as the more stock in storage, the more there is available to meet supply and therefore the undersupply value will be lower.

Note One of the major disadvantages with the simulated annealing algorithm is that it is very sensitive to the choice of initial model parameter values. This disadvantage is very much present in the application of the simulated annealing algorithm to the real world stochastic simulation model. An example is used to demonstrate this disadvantage.

The simulated annealing algorithm was implemented the same way as described above. As defined in the method, the initial model parameter values are randomly selected from within the defined constraints for that parameter. Table 4.6 shows the goal value, as well as the objective values, of the initial model as well as the final goal and objective values achieved for two runs of the simulated annealing algorithm; the one defined above, called run SA1 and an other, called run SA2. From this it can be seen that the final goal value of run SA2 is not even as good as the goal value of the initial model used for run SA1. Figure 4.9 shows the convergence path that the goal value for run SA2 takes. From this it can be seen that the simulated annealing algorithm is working well in terms of the fact that worsening moves are accepted towards the beginning of the algorithm but due to the initial parameter values, convergence tends to take

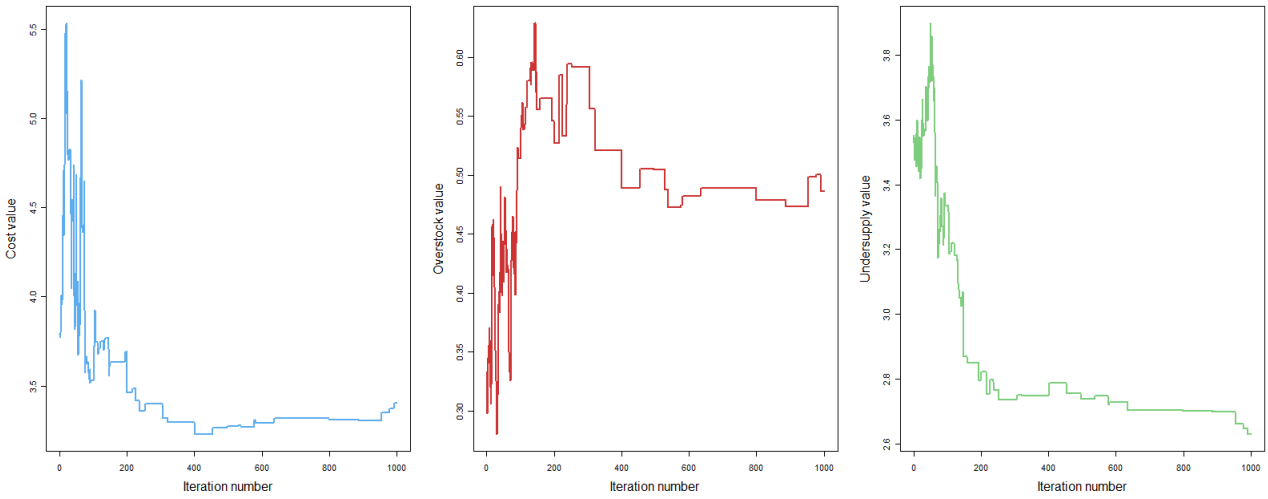

Figure 4.8: Simulated annealing objective value convergence patterns

place at a local optimum.

A way in which this disadvantage could be overcome would be to increase the run length but then this raises the issue of cost versus accuracy - a better solution might be found if the algorithm was left to run for a longer period of time but this would then increase the computational run time of the model. Another way would be changing the temperature to avoid local parameter convergence, but this in itself, would also require large computational time in terms of testing the different temperature values that would be used. A few runs were performed with a higher temperature value of 100 but that did not seem to create any significant improvement in the results except that through increasing the temperature, more worse moves were accepted for a longer period of time as the Boltzmann probability took a longer time to reach 0.

Run	Initial Model				Final Model			
	Goal value	Cost value	Over stock value	Undersupply value	Goal value	Cost Value	Over stock value	Undersupply value
SA1	25.107	1343032	2256	19864	18.278	1152559	2083	5179
SA2	33.282	886139	1000	65792	26.881	795091	1460	48694

Table 4.6: Initial and final values for two runs of the simulated annealing algorithm

Conclusion

After having decided the specifications required for the simulated annealing method to be implemented, it was run on the real world stochastic simulation model and a good final goal value was achieved. It is noted that this goal value achieved is the best value generated from implementing the simulated annealing algorithm 30 times. This indicates that that the simulated annealing algorithm is very dependent on the parameter values of the initial model that is used

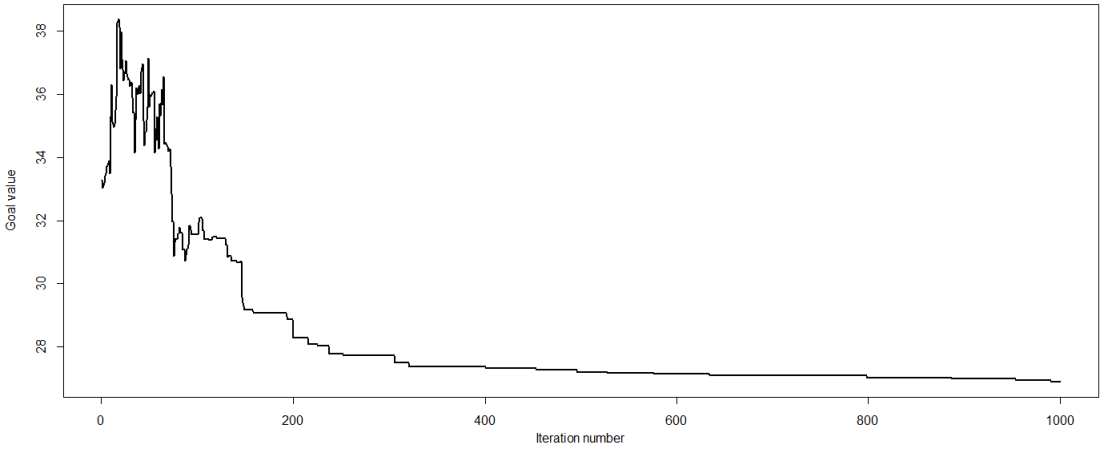

Figure 4.9: Run SA2 goal value convergence pattern

at the start of the method process. On closer evaluation of the objectives that make up the goal, it was confirmed that the overstock objective and undersupply objective are inversely related but due to the relative importance of the objectives, a percentage change in undersupply was more significant on the goal value than a percentage change in overstock.

From the results that were generated this method appears a promising one for optimising the real world stochastic simulation model.

4.2.3 Genetic Algorithm

The next simulation optimisation technique that is implemented on the real world stochastic simulation model is that of genetic algorithm. As previously described, this method is a heuristic optimisation method that models the natural evolution process. The same terminology that was used when applying the genetic algorithm to the test case simulation model is used for this case.

Preliminary Information

Before the genetic algorithm simulation optimisation technique can be implemented, certain preliminary decision need to be made. This is done to tailor the genetic algorithm to the simulation model that is to be optimised. The decisions that were required are explained in more detail below. It is noted that the decisions were made using the knowledge gained from implementing the genetic algorithm optimisation technique to the test case simulation model but that should the genetic algorithm technique prove to be a useful and good method to implement, further revision of these decisions could be made.

- Initial population creation
  - The initial population forms the base on which future solutions are built and is created by randomly selecting sample designs from the feasible space. This means

that each parameter value is randomly selected from the range over which they are defined and in such a way that the constraints are satisfied. This means that there are a great number of possible solutions that can be part of the initial population.

- Population size
  - The population in each generation is made up of two aspects: the parent solutions and the children solutions. For this case, the number of parents was set to be equal to the number of children. This means that in one generation, there are 50 parent solutions. From these 50 parent solutions, 50 children solutions are created and together that makes a population size of 100. When the next generation is created, only 50 solutions from the current generation will move onto the next generation, in turn creating the 50 parent solutions. The reason that each generation was of size 50 and not 100 like in the test case was due to the fact that this method takes a long time to implement. Again, if promising results were generated the population size could be explored.
- Type of string coding that will be used for the parameters
  - As previously explained, the genetic algorithm does not use the parameter values themselves but rather a string representation of the parameter values. This is done so that the required stages of the genetic algorithm method of mutations and crossovers of the parent parameter values to generate the children parameter values can take place. The type of string encoding that is used is binary string encoding. This means that the parameter values are broken down into 0s and 1s. These strings can the be decoded to get the actual parameter values.
  - There are 4 types of parameter values that have to be defined to run the simulation model. These refer to the quality of the stock, the amount of stock delivered, the planned maintenance and lastly the cost of supply allocated to each plant. Due to the way in which these parameters were defined, the string coding had to be adapted to suit each of these parameters.
  - The quality of the stock and the amount of stock delivered to each plant are both defined on a continuous scale. As such, normal string encoding is used for these variables (as seen in the test case simulation model example). It was decided that a 10 bit encoding for these two variables would be used. This meant that for each variable, there are a possible of 1024 solutions that can be generated. The reason that 10 bits encoding was used was due to the fact that there is a large number of parameters with a large range of possible values. As such, by having a smaller bit number means that there are a smaller number of possible combinations of solutions which is desirable as this ensures that the probability of all solutions coming from the same area of the feasible space is small. In order to decode the parameter value

the following formula was used:

$$x_i = x_L + \frac{\sum_{i=0}^{bits-1} 2^i a_i}{2^{bits} - 1} \times (x_U - x_L)$$

Where x_i is the decoded parameter value, x_L is the lowest value that the parameter can take, x_U is the highest value the parameter can take, bits is the number of bits that the parameter has (10 in this case) and a_i is the bit value in the ith position (either 0 or 1) [47].

- For the cost of production parameter, binary string encoding cannot be used due to the way in which the parameter is defined. The cost of production variable takes on a discrete value from 1 to 13 and no two plants can have the same value (unless it is plant J and K). As such, no encoding for these parameters takes place and the cost values themselves are used.
- Due to the way that the planned maintenance is defined, it was decided not to use string encoding for these parameters. This is due to the fact that for the planned maintenance, the yearly total maintenance for each plant needs to be defined to a specific value. As such, the parameter values are left as is. To define these parameter values for each plant the following takes place; the yearly planned maintenance needs to be allocated to each of the 12 months. To do this 12 randomly uniform values are generated of which the sum is taken. These 12 values are then normalised by dividing each one by the sum. These normalised values are then multiplied by the yearly planned maintenance value for that plant to give the percentage that the plant needs to be shut down for maintenance. An example of this is shown for plant A in Figure 4.10.

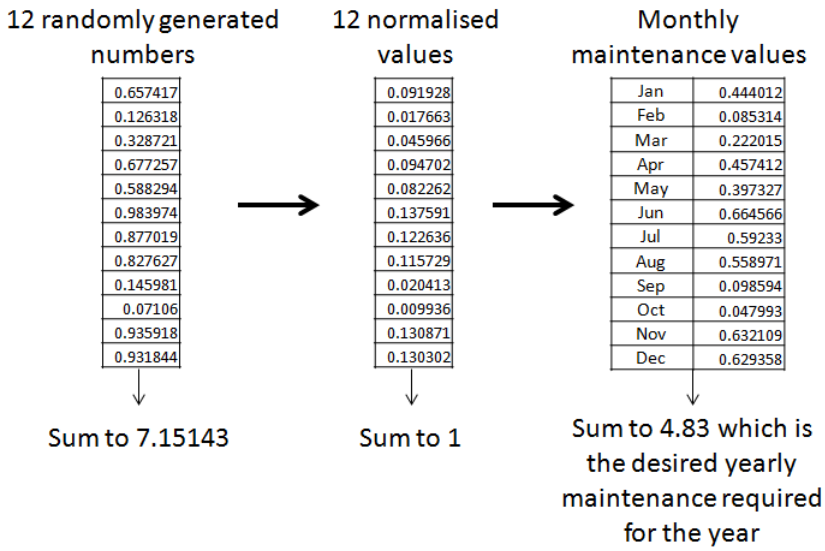

Figure 4.10: Example of planned maintenance value generation for plant A where the yearly planned maintenance is required to be 4.83%

- Children creation and crossover mechanism
  - At each generation, 50 children are generated from the 50 parent solutions in that generation. These children are generated as follows until all 50 children have been created:
    1. 4 parent solutions are randomly selected
    2. From these 4 individuals, the best performing 2 are selected
    3. These 2 individuals are then crossed over to create a child
  - For each parameter, the crossover mechanism is defined in different ways. The crossover operation for the delivery parameter and the quality of stock takes place by taking the binary string representation for each of the parents. If the parent has the same bit value at the same position in the string then the child gets that bit value at that position in its sting. If the parents have different bit values at the same point in the string representation then a 0 or a 1 is randomly generated and takes the corresponding position in the child's binary string representation.
  - A similar approach is used for the cost of production variable. For this, if the plant in both of the parents is allocated the same cost value then the child gets that cost value. If the cost value for the plant varies between the two parents then the child gets a randomly generated cost value that has not yet been allocated. An example of this is shown in Figure 4.11.

	Cost allocated to plant:													
	A	B	C	D	E	F	G	H	I	J	K	L	M	N
Parent 1	5	11	3	4	6	10	2	12	1	7	7	8	13	9
Parent 2	6	11	7	4	12	10	1	2	3	5	5	9	13	8
Chlid	3	11	5	4	7	10	6	12	8	2	2	9	13	1

Figure 4.11: An example of a child cost variable being generated from two parents

  - For the planned maintenance, the values for each month for each plant for both the parents were compared. If the parents had the same planned maintenance value (which is achieved with essentially 0 probability do to the number of possible maintenance values) then the child gets the same value as both the parents' planned maintenance percentage for that plant at the month. To get the planned maintenance for the remaining months of each plant in the child solution, the same approach that was used to create the planned maintenance values was used but only the months that had yet to be given values in the child were defined. An example, again for plant A where the planned maintenance for the year is 4.83%, is given in Figure 4.12. It is worth noting that for future GA development, a different approach might be taken when generating the children for planned maintenance. This could be done by taking two parents as above, except that this time the planned maintenance value

for the child would be the average of the two planned maintenance values of the parent values. Through this it would mean that if the planned maintenance value of both the parents were relatively large then the planned maintenance value of the child would also be large and similarly if the two plannned maintenance values of the parents were relatively small then the child planned maintenance value would also be relatively small. This might ensure that more of the genetic information is carried through from one generation to the next.

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Parent 1	0.7597	0.5416	0.5266	0.3236	0.1919	0.0719	0.8181	0.7755	0.0699	0.1575	0.3006	0.2931
Parent 2	0.7597	2.0458	1.9889	1.2225	0.7250	0.2716	3.0901	2.9291	0.2641	0.5950	1.1356	0.2931
Child	0.7597	0.1990	0.4505	0.4770	0.4653	0.2348	0.4285	0.3255	0.3529	0.4618	0.3819	0.2931

Figure 4.12: An example of a planned maintenance for a year for plant A being generated from two parents

- The mutation rate and point at which the mutation rate takes place.
  - The mutation occurs on the children parameter values to prohibit getting stuck at a local optimum.
  - The mutation rate is set to 0.1. This means that 5 of the 50 children will get mutated. These 5 solutions are randomly selected.
  - Again, due to the way in which each of the parameter values are defined, the mutation rate is different for each of the parameters.
  - For the quality parameter, each solution has 14 quality values (each plant has their quality value that remains constant throughout the year period). To ensure that not all the quality values within that solution are mutated only 50% (7) of the parameter values are randomly selected. From this, one bit value of each of these parameters is randomly selected and is mutated in the sense that if the bit value is 0 it becomes 1 and if the bit value is 1 it becomes 0.
  - Similarly, for the delivery of stock parameter, each solution has 168 delivery values. Again, 50% (84) of the parameter values are randomly selected to undergo mutation. Once these parameters are selected, 1 bit value for each parameter is randomly selected to be mutated.
  - For the cost parameter, there are a total of 14 values (1 value per year for each plant). The mutation for this parameter takes place by randomly selecting two plants and swapping their cost values. It is noted that the two plants that have to have the same cost value by definition are treated as 1 plant when this occurs to ensure that their constraint of their having to have the same cost is always satisfied.
  - Lastly, for the planned maintenance parameter, a similar approach is taken that was used to generate the child solution from the parent solution. There are 168

maintenance values (one for each month for each plant). To carry out the mutation, 7 out of 14 plants are randomly selected for mutation. Then, for each of the plants selected, 2 maintenance values are selected and their sum is calculated, call this m. Two uniform values are randomly generated, they are normalised and then multiplied by m and this defines the new maintenance values for those two months and ensures that the constraint is satisfied.

  - It is noted that in future GA development, when performing the mutation, that a proportion of the genes in every child are mutated instead of a proportion of the children might be performed to see if this allows the GA method to perform better.

- Selection mechanism

  - The selection mechanism is responsible for selecting the individuals from the current generation that move to the next generation to become the parents of that generation. The selection mechanism used is the elitist selection mechanism. This works by selecting the 50 best performing individuals. This means that the next generation will be made up of the top 50 solutions of the the current generation.

- Stopping criteria and optimal value selection

  - The stopping criteria that was used for this simulation optimisation model was to stop the process once a desired number of generations had been reached. It was decided that this would take place once 50 generations had taken place.

Simulation Optimisation

Now that all the preliminary information has been defined, the genetic algorithm procedure can be implemented. The following steps take place to do this:

1. Randomly generate the 50 parent solutions using the string coding previously explained and decode any parameters that require to be decoded.
2. Run the simulation model using these 50 parent solutions.
3. Generate the 50 children solutions using the above defined crossover mechanisms.
4. Apply the mutation to the selected children as previously defined and decode the parameter values that are required to be decoded.
5. Run the simulation model for the 50 children solution.
6. Select the best 50 solutions from the generation to create the parents for the next generation.
7. Repeat the process from step 3 until the desired stopping criteria has been met.

8. The optimum solution is defined as the solution that has the lowest goal value in the final generation.

Figure 4.13 shows the goal values for the solutions in the final generation after the genetic algorithm has been run. These values have been ranked from lowest to highest. From this it can be seen that the best goal value that is generated is 31.484. Table 4.7 shows the values that the three objectives have. It is noted that when comparing this goal value to that generated by the various alternatives in the previous section, it appears as though this final goal value achieved by the genetic algorithm technique is large and is definitely not at a minimum. This is an indication that the genetic algorithm method does not work well for this real world stochastic simulation model. Looking at the objectives that make up the goal, it is seen that the large goal value is due to the large amount of overstock. This is confirmed when comparing the value of the overstock objective generated to the ideal overstock value.

Goal value	Cost value	Overstock value	Undersupply value
31.484	1319247	13969	2169

Table 4.7: Final solution from genetic algorithm - goal and objective values

Upon closer evaluation, it is noted that this particular implementation of the genetic algorithm is not well suited to the simulation optimisation problem due to the fact that the solution generated does not produce a good result. When looking at the objectives in turn, it was especially evident that the overstock objective was not being minimised correctly by this method and therefore resulted in such a large goal value being produced. The main reason that this implementation of the method was not successful was due to the combination of the fact that the feasible solution space is large and that there were many continuous parameters that struggled to handle the binary encoding. It was seen that although having a small bit number ensured that there were fewer possible solutions for each parameter and therefore increased the probability of the whole search space being explored, it proved to be a problem as the solution was limited in terms of how well it could be refined as certain parameter values can never be explored due to the bit encoding. This is because the bit encoding segments the feasible solutions space and as such the possible values are discrete ones based on the segments and not continuous values.

Due to the results generated by using the above specifications and comparing them to the results generated by the other simulation optimisation techniques, it was seen that the optimum goal values that the genetic algorithm method obtained were very much higher than those of the other methods. As a result, it was decided to no longer pursue this method by trying to change the preliminary specifications to see if that would have an effect on the solutions generated. It is noted, however, that there a different implementation of this method might lead to more satisfactory results.

Conclusion

The genetic algorithm simulation optimisation technique was applied to the simulation model and the optimum result was calculated. It was seen that the results achieved by this method were not very good and upon further analysis it was seen that a factor that could be the cause of this is that the overstock objective was very high and its minimising was difficult due to the way in which the genetic algorithm works.

It is noted that potential improvements when applying this method might be to only select one of the parameters and make one (or more) mutations to the single parameters instead of all at once as was done in this case.

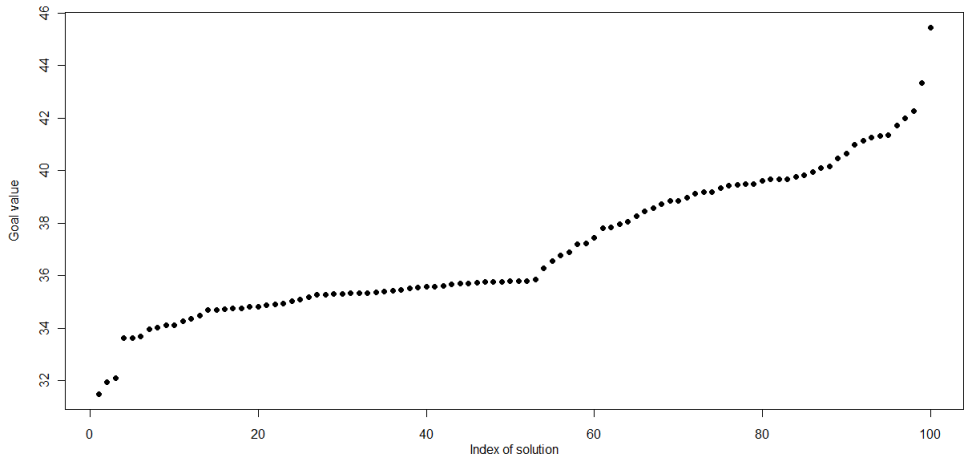

Figure 4.13: Goal values generated by the final generation of the genetic algorithm

4.2.4 Ordinal Optimisation

The ordinal optimisation method was then tested on the real world stochastic simulation model. This method takes a goal softening approach and focuses on the order of the goal values rather than the goal values themselves by using a crude simulation model instead of the accurate simulation model. Figure 4.14 demonstrates how ordinal optimisation quantifies the 'narrowing down' of the search space with guaranteed high probability and goodness. Again, the same procedure is applied when applying this method to the real world stochastic simulation model as when this method was applied to the test case simulation model. It is noted that although this method was shown not to perform very well on the test case simulation model it is still implemented on the real world model to see if the results generated would be better.

Preliminary Information

One of the major advantages of the ordinal optimisation method is that no prior knowledge of the simulation model is needed and the method does not need to be adjusted to suit the simulation model (such as, for example, in simulated annealing where the neighbourhood region needed to

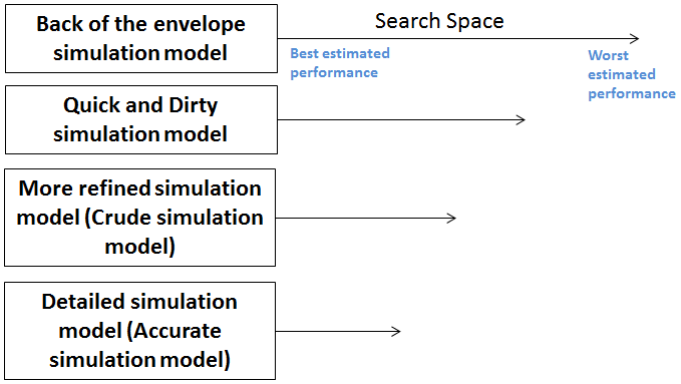

Figure 4.14: Reduction of search space through ordinal optimisation

be defined or in genetic algorithm where the number of bits had to be selected). As a result, no decisions need to be made prior to the method being implemented.

Simulation Optimisation

The OO process is now implemented on the real world stochastic simulation model and is explained below. The aim of this method is to get a group of sample designs that contain k truly good enough designs with a probability level no less than the defined alignment probability.

Step 1 The first step of the process requires that a sample of possible solutions are randomly selected from the feasible space. A possible solution was generated by randomly selecting the value for each of the 700 parameter values. Using the same approach as that outlined in [65] and in the test case simulation model example, a sample a 1000 solutions are selected from the feasible space. This selected sample is defined as Θ.

Step 2 The next step is to estimate the performance of these 1000 selected designs using a crude and computationally fast model. A crude model can either be a simplified version of the current simulation model or it can be the current simulation model run for a shorter period of time. For the purposes of the real world stochastic simulation model, since it was created by an outside source and therefore complete understanding of the intricacies of the model were not known, it did not seem advisable to create a simpler version of the simulation model as vital information may not be correctly carried through to this simpler simulation model. Instead, what was done was to cut the number of iterations of the real world stochastic simulation model by 50% to get the crude model. This means that the crude model was the real world stochastic simulation model run for half the number of iterations. The running time of one run of the real world stochastic simulation model is 46 seconds and the run time of the crude simulation model is 31 seconds and as a result, the running time has now decreased by 33%.

The 1000 samples selected to make Θ in step 1 are then run using this crude simulation model and their goal value is recorded. The 1000 goal values are then ordered in ascending order

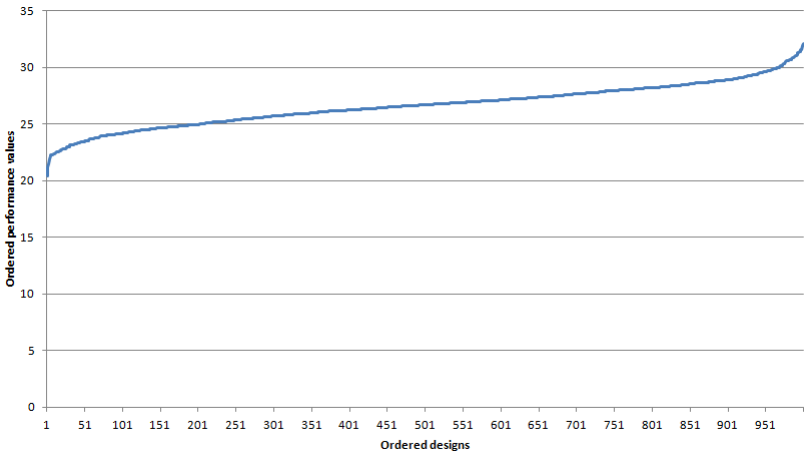

Figure 4.15: Ordered performance curve for output of the crude model associated with the real world stochastic simulation model

so that an ordered performance curve (OPC) can be drawn. The shape of the OPC determines the nature of the underlying optimisation problem. The OPC derived from the output of the 1000 samples is shown in Figure 4.15. From this it can be seen that the OPC shape is bell shaped which means that there are many intermediate schemes.

Step 3 Certain values need to be defined in order for the OO process to carry on; the size of the good enough subset, the required alignment level, the corresponding alignment probability and the error bound. These are defined as follows:

- The good enough subset G is defined as the top 5% solutions of Θ^N, thus $g = N \times n\% = 1000 \times 5\% = 50$
- The alignment level $G \cap S$ is $k = 1, 2, 3, 4$ or 5
- The alignment probability of $G \cap S$ is $p = 95\%$
- The error bound is then calculated as described below using the method described in [43]. The error bound is necessary in order to calculate the size of the selected subset.
  - 30 samples are randomly selected from the 1000 samples
  - These 30 samples are run using the real world stochastic simulation model and their goal values are calculated and recorded
  - From the output, the variance of the normalised approximation error is calculated as 0.0047.
  - The error bound, W, which is used to define the distribution $U[-W, W]$ is then calculated as

$$\text{variance of the uniform distribution} = 0.0047 = \frac{1}{12}(2W)^2$$

$$W = 0.119$$

The value is rounded to be equal to 0.5. This is due to the fact that the error term, W, is defined for 0.5, 1 and 2.5 as shown in [65].

Step 4 The size of the selected subset is now calculated. Below is a summary of all the information that is required to calculate the size of the selected subset. Using this information and Table 3.20, shown in the ordinal optimisation section of the test case simulation model, the size of the selected subset can be defined and is shown in Table 4.8.

- Size of the representative subset $N = 1000$.
- The OPC is bell shaped.
- The good enough subset G is defined as the top 5% solutions of Θ^N, thus $g = N \times n\% = 1000 \times 5\% = 50$.
- The alignment level $G \cap S$ is $k = 1, 2, 3, 4$ or 5.
- The alignment probability of $G \cap S$ is $p = 95\%$.
- The error bound is defined to be 0.5.

	OPC shape
g = 50	Bell
k=1	12
k=2	15
k=3	21
k=4	29
k=5	39

Table 4.8: Size of selected subset for real world stochastic simulation model

Step 5 and 6 The size, s, of the selected subset for the various values of k has now been determined. The top s designs for each value of k are then selected (based on the output given by running the samples through the crude model) and these designs are run using the real world stochastic simulation model and the goal is calculated and is recorded. It is noted that OO focuses on order and not value and therefore, the design is defined as best if its goal value is ranked to be the best. However, it is still worthwhile in looking at the goal value achieved by the best design in order to see how well this method performs, especially when comparing it to the other methods. Table 4.9 shows the goal value of the best design found, having the lowest goal value, when the OO is completed. This design was also ranked first when producing the OPC using the crude model. This gives an idea of the goal value but not necessarily that this is the optimum value.

Now that the samples in each subset have been run with the real world stochastic simulation model, we now have selected subsets that have at least k designs in G with a probability of 95%.

Goal value	Cost value	Overstock value	Undersupply value
20.338	1171120	12886	1802

Table 4.9: Best solution found by the ordinal optimisation method - goal and objective values

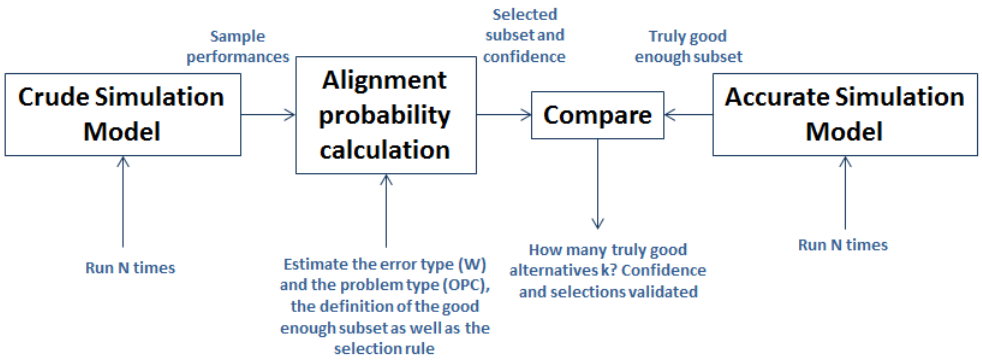

Figure 4.16: Implementation and validation of ordinal optimisation method

Further analysis can then be conducted on these samples to get a better understanding of their make-up. If the process were to be stopped here there would be no knowledge as to which of the samples in the selected subset represent the at least k best designs. For purposes where the simulation models are very time consuming to run the OO procedure would be stopped here. For the purposes of this study however, the real world stochastic simulation model is not too lengthy to run and therefore OO validation is performed.

An overview of the implementation and validation of OO is shown in Figure 4.16.

Validation of Process The validation of the OO approach and its corresponding results requires the good enough subset be calculated. This is done by running all the 1000 samples using the real world stochastic simulation model and selecting the top 50 designs. As already explained, if alignment level is set to be k, then of the s designs making up the selected subset S, at least k schemes will fall in the good enough subset G with the probability of 95%.

Table 4.10 shows the good enough subset and the selected subset for various values of k. The numbers in the table represent the index number allocated to the design when it was randomly generated to create Θ. In this particular case it is seen that the number of elements in $S \cap G$ is much higher than the required alignment level k. The results of this case are much better than expected; all elements of S fall into G, however, with different order rankings. Upon closer evaluation, it was seen that the reason for this was that the crude model as well as the actual model produced very similar results when the model was run. This indicates that decreasing the simulation run time, as done here, does not seem to affect the results generated. This result might not hold true if the simulation run time is decreased further.

k value	S size	Size of $S \cap G$	Selected Subset	Good Enough Subset
1	12	12	977, 42, 946, 559, 311, 222, 317, 300, 756, 954, 290, 455	**977**, **42**, **946**, **559**, **311**, **317**, **222**, **300**, **290**, **756**, **954**, 381, 425, 208, 275, **455**, 116, 583, 176, 435, 234, 824, 284, 64, 336, 168, 174, 652, 519, 545, 76, 757, 393, 606 404, 814, 488, 730, 906, 721, 183, 34, 6, 389, 5, 588, 707, 325, 334, 555
2	15	15	977, 42, 946, 559, 311, 222, 317, 300, 756, 954, 290, 455, 208, 381, 116	**977**, **42**, **946**, **559**, **311**, **317**, **222**, **300**, **290**, **756**, **954**, **381**, 425, **208**, 275, **455**, **116**, 583, 176, 435, 234, 824, 284, 64, 336, 168, 174, 652, 519, 545, 76, 757, 393, 606 404, 814, 488, 730, 906, 721, 183, 34, 6, 389, 5, 588, 707, 325, 334, 555
3	21	21	977, 42, 946, 559, 311, 222, 317, 300, 756, 954, 290, 455, 208, 381, 116, 425, 583, 275, 176, 435, 234	**977**, **42**, **946**, **559**, **311**, **317**, **222**, **300**, **290**, **756**, **954**, **381**, **425**, **208**, **275**, **455**, **116**, **583**, **176**, **435**, **234**, 824, 284, 64, 336, 168, 174, 652, 519, 545, 76, 757, 393, 606 404, 814, 488, 730, 906, 721, 183, 34, 6, 389, 5, 588, 707, 325, 334, 555
4	29	29	977, 42, 946, 559, 311, 222, 317, 300, 756, 954, 290, 455, 208, 381, 116, 425, 583, 275, 176, 435, 234, 64, 824, 284, 168, 336, 652, 519, 174	**977**, **42**, **946**, **559**, **311**, **317**, **222**, **300**, **290**, **756**, **954**, **381**, **425**, **208**, **275**, **455**, **116**, **583**, **176**, **435**, **234**, **824**, **284**, **64**, **336**, **168**, **174**, **652**, **519**, 545, 76, 757, 393, 606 404, 814, 488, 730, 906, 721, 183, 34, 6, 389, 5, 588, 707, 325, 334, 555
5	39	39	977, 42, 946, 559, 311, 222, 317, 300, 756, 954, 290, 455, 208, 381, 116, 425, 583, 275, 176, 435, 234, 64, 824, 284, 168, 336, 652, 519, 174, 545, 906, 393, 76, 404, 814, 606, 5, 488, 730	**977**, **42**, **946**, **559**, **311**, **317**, **222**, **300**, **290**, **756**, **954**, **381**, **425**, **208**, **275**, **455**, **116**, **583**, **176**, **435**, **234**, **824**, **284**, **64**, **336**, **168**, **174**, **652**, **519**, **545**, **76**, 757, **393**, **606 404**, **814**, **488**, **730**, **906**, 721, 183, 34, 6, 389, **5**, 588, 707, 325, 334, 555

Table 4.10: Ordered designs for the selected subset and the good enough subset for different levels of k for the real world stochastic simulation model

Conclusion

Using the OO approach demonstrates that it is an effective simulation optimisation technique in the sense that it is able to pick a selected subset in which good enough solutions can be found with high probabilities and therefore further effort in the future can be confined to the elements of a small selected subset. This is particularly useful for this real world stochastic simulation model due to the fact that the feasible space is so large. It was also seen when drawing the OPC curve that there appear to be many intermediate solutions that deliver intermediate goal values. It is also seen to be an effective simulation optimisation technique due to the fact that the minimum goal value achieved by this method is good compared to that achieved by the other simulation optimisation techniques tested. The goal value achieved by this method is not as good as the one achieved by the simulated annealing technique but is much better than that achieved by the genetic algorithm.

By retreating from focusing on nothing but the 'best' solution and treating the optimisation with a goal softening approach such that focus is given on order and not value and emphasis is placed on having a good enough solution then a lot of saving can be made in terms of time it takes to run the simulation optimisation technique. Therefore, in conclusion, this method has produced not one 'best' solution but rather a group of solutions that are ranked in the top 5% of solutions and who perform quite well in terms on minimising the goal value. It is noted that the quality of the solutions could be improved if more than 1000 solutions for the feasible space were selected in step one.

This method is a good method to apply to the real world stochastic simulation model to get a good group of solutions with good goal values in a short amount of time but does not produce the optimum result.

4.2.5 Nelson-Matejcik Method

Nelson and Matejcik established a fundamental connection between indifference zone R&S procedures and MCB. The idea of combining indifference zone R&S approaches with MCB is appealing to the simulation analyst. Such an approach not only selects the best system with prespecified confidence, but it provides inferences about the relationships between systems which may facilitate decision making based on secondary criteria that are not reflected in the performance measure selected.

Preliminary Information

The Nelson-Matejcik (NM) method was then applied to the real world stochastic simulation model using the same steps that were used when applying this method to the test case simulation model. The aim in applying this method is to minimise the goal value which is a multi-objective goal made up of excess stock on hand, cost of production and undersupply penalty.

The major advantage with this method is that prior knowledge of the simulation model and how it is run is not needed. This is particularly significant with this model as, unlike the test

case, it was constructed by an outside source and therefore the intricacies are not as well known. As a result, no preliminary steps need to be taken prior to the implementation of the method.

Simulation Optimisation

The NM method is now implemented on the real world stochastic simulation model and each step is explained in detail. It is noted that although this method performed poorly in the test case simulation model, it is implemented on the real world stochastic simulation model to see if perhaps it might before better on this model.

Step 1 As previously explained, there are a total of 700 parameters whose values need to be decided upon to produce the optimum result of minimising the goal. These parameters are divided among the quality of stock, the planned maintenance, the stock delivery and the cost of production and each has either a specific allowable range or is defined in a specific way. There is therefore a large number of possible feasible solutions which could result in the minimum goal value (specifically since some of the parameters are defined over a continuous range). Using the same approach used when performing this method of the test case simulation model, the number of possible solutions selected from the feasible space is 1000. This sample is randomly selected from the feasible space and is defined as $\hat{\Theta}$. This random selection is done so that each time a new feasible solution is created, each of that variable values are randomly selected from their range.

The aim of the NM procedure will then be to decide which of these feasible solutions results in the optimum cost value. It is noted that since the NM procedure is performed on a sample of the feasible space that contains at least one design that is in the top 5% of the feasible space with a probability of 0.95, the optimum result generated by this process therefore is defined not as the optimum design but as a design present in the top 5% of designs.

Step 2 Step 2 of the NM method is to ensure that the output data (the goal value) is normally distributed throughout the simulation run. The way that this is normally done is to run the simulation model using a baseline case for a certain number of iterations and recording the goal value at the end of each iteration. From this, the method of batched means is used to estimate the steady-state mean and/or variance of a simulation output measure which is then used when performing tests of normality to see if the output measure is in fact normally distributed, and if so the method can proceed. This has been shown when applying the NM to the test case simulation model in the previous chapter.

Due to the way that the real world stochastic simulation model is created, there is no way in which the goal value at the end of each simulation model can be saved. This is because the real world stochastic simulation model is essentially a black box as it takes inputs in, runs the model and then only produces outputs at the end of the model run and does not produce any measures whose values are tracked throughout the simulation model run. As a result of this fact, there is no way to carry out any tests of normality on the data without compromising the simulation results.

Although no normality tests can be performed, this is not seen as a hindrance, however, as it has been proven by Nelson and Matejcik [76] that this process is extremely robust to departures from normality and therefore, although we are unable to tell if the goal generated from the simulation model is normally distributed throughout the model run, we are still able to implement the NM method on the real world stochastic simulation model. Therefore, although this step is carried out generally when performing NM, it is not necessary to be able to proceed with this method.

Step 3 In this step, a confidence interval is created based on the goal values generated for each of the samples in $\hat{\Theta}$. To do this, the simulation model is run for each sample in $\hat{\Theta}$ using common random numbers and the goal value is recorded each time. Common random numbers are used so that each sample is run using the same number configuration and serves as a variance reduction technique. At the end of this process, there are 1000 goal values generated after having run the simulation model 1000 times.

A 95% confidence interval is then built using these 1000 goal values and calculated as [26.504; 26.735]. From this, all the samples that generated a goal value of less than 26.735 were selected as the sample on which further analysis will be performed. The reason for the upper bound being selected is that the optimisation problem is a minimisation one. This size of this new sample is 515 and will be refered to as the final sample for the remainder of the procedure.

The reason that this step is carried out it to reduce the size of Θ by eliminating solutions that are not optimal so that the method can continue with a smaller sample size and therefore complete in a shorter amount of time. The reason that the solutions that are lower than the top bound of the confidence interval are kept is since there is simulation noise there is a chance that the optimal solution did not perform in the best way on this run of the model. By keeping this group and running further the model more times below, this simulation noise will be accounted for and thereby help identify the optimum solution of the sample.

Step 4 In order for the next step to be performed, certain values need to be specified and are explained below:

- $\alpha = 0.05$. By selecting this value, the probability of correctly selecting the optimum design is 0.95.
- The number of observations to collect during the first stage of the procedure = 30.
- From the above and the fact that the final sample size is 515, the next value that is calculated is g where $g = T^{1-\alpha}_{k-1,(k-1)(n_0-1),0.50}$ is the $(1-\alpha)$-quantile of the maximum of a multivariate t random variable with $k-1$ dimensions, $(k-1)(n_0-1)$ degrees of freedom and common correlation 0.50. This is obtained through the tables in [52] and is equal to 2.68.
- The indifference zone, $\delta = 1$. This means that two samples that have goal values that differ by less than 1 are indifferent to one another.

In order to come up with a way to define the indifference zone, the goal value had to be looked at a little more closely and deconstructed based on the objectives. If the goal value represented a cost, for example, it would be easy to specify the indifference zone as a value can easily be found which is the indifference zone. Since for this case the goal is a multi-objective one, specifying the indifference zone is more complicated. This is due to the fact that the goal value itself has no actual meaning. Therefore, just looking at the goal value to come up with the indifference zone is not possible. To come up with the indifference zone the following steps were taken:

1. It was decided that to calculate the indifference zone, the alternative analysis would be incorporated as the alternative analysis was used to provide insights into the simulation model.

2. From the alternative analysis it was seen that alternative D produced the best goal result of 20.68 units and is the alternative selected to help construct the indifference zone.

3. Using the solution generated by alternative D as a reference point, the 3 objectives making up the goal were taken in turn and were all decreased by 5% and their new values were recorded. Call this model 2. This was done as it was decided that any objective value that differed by 5% or less to the current objective value were indifferent to one another. Although each objective was decreased by 5% it is noted that based on objectives only this creates objectives that are indifferent but it might not be feasible to create these objective values due to the constraints that are in place in the simulation model. It is noted at this point that 5% is not necessarily the correct value but is chosen as a simplifying assumption.

4. The difference between the objectives for alternative D and model 2 was then calculated and these values were used as the indifference values for each objective. This means that for the cost of production objective, objectives that differed by 57929.82 or less were indifferent to one another, for the yearly overstock objective, solutions that differed by 13.74 or less were indifferent to one another and lastly, for the undersupply objective, solutions that differed by 879.70 or less were indifferent to one another.

5. The goal was then calculated manually using different combinations of the objective values from alternative D and model 2. The effect that these combinations have on the goal are shown in Table 4.11. The indifference zone is then taken to be the difference between the maximum goal value in the table and the minimum goal value in the table. This is defined as 0.99 and results from the difference between the all objectives having alternative D values and all the objectives having model 2 values, as one would expect.

6. The indifference value is then set to be 1. This means that if two solutions differ by 1 unit or less, they are considered indifferent to one another.

It is worth noting that the values defined in 4 above act only as guidelines to come up with the value of the indifference zone. Two models might be indifferent to each other even if the

Combination of objectives			Goal value
Cost of production	Yearly over stock	Yearly undersupply	
Alternative D	Alternative D	Alternative D	20.685
Alternative D	Alternative D	Model 2	20.477
Alternative D	Model 2	Alternative D	20.676
Model 2	Alternative D	Alternative D	19.941
Alternative D	Model 2	Model 2	20.438
Model 2	Alternative D	Model 2	19.704
Model 2	Model 2	Alternative D	19.932
Model 2	Model 2	Model 2	19.694

Table 4.11: Combination of objective values to calculate indifference zone

absolute difference between the objective values for the two models are larger than the values defined above. This is because the indifference zone is defined for the difference in goal values and not the difference in objective values.

Step 5 In the previous step, the number of initial replications of each sample in the final sample was set to be 30. The simulation model is then run, for each of these samples, 30 times. It is noted that common random numbers are used here again. This means, for example, that when run number 2 is performed, all the samples are run using the same configuration of random numbers and then again, for run number 3, all the samples are run using the same random number configuration but this configuration is different to the one used in run 2 etc. At the end of each run for each sample the goal value is calculated and saved. It is noted that a disadvantage of this method is evident at this point and that is the time it takes to run all the 515 designs 30 times. Although the simulation model itself does not take very long to run, placing the input values in the correct format, running the simulation, capturing the output and then calculating the goal value does take time.

From these goal values, the sample variance is then calculated. The sample variance is defined as

$$S^2 = \frac{2\sum_{i=1}^{k}\sum_{j=1}^{n_0}(Y_{ij} - \bar{Y}_{i\cdot} - \bar{Y}_{\cdot j} + \bar{Y}_{\cdot\cdot})^2}{(k-1)(n_0-1)}$$

where Y_{ij} is the j^{th} replication of the i^{th} design and $\cdot$ represents averaging with respect to that subscript. The sample variance for the 30 replications of the 515 designs is equal to 0.0026.

Step 6 The next step is to calculate the final number of replications required for each design as shown below:

$$\begin{aligned} N &= max[n_0, (gS/\delta)^2] \\ &= max[30, 0.0187] \\ &= 30 \end{aligned}$$

As a result, it can be seen that N is equal to 30. Since the 30 initial replications have already been performed, no additional replications need to take place and the remaining steps are

performed on the 30 goal values for each of the samples in the final sample.

Step 7 Now that all the required replications have been performed for all the samples in the final sample, the mean goal value is calculated for each sample. This is calculated as

$$\bar{Y}_{i\cdot} = \frac{1}{N}\sum_{j=1}^{N} Y_{ij} \text{ for } i = 1, 2, ..., k$$

The design that has the smallest overall sample mean is selected as the best design. This is the design that has a goal value of 20.347. From a R&S perspective this means that with a probability of 0.95 that this design lies within $\delta = 1$ of the true best.

Step 8 The last step of the process is to calculate the confidence intervals. This is done to see if the output value from the other designs are within the indifference zone. The confidence intervals are calculated using the following formula:

$$\mu_i - \min_{j \neq i} \mu_j \in [-(\bar{Y}_{i\cdot} - \min_{j \neq i} \bar{Y}_{j\cdot} - \delta)^-, (\bar{Y}_{i\cdot} - \min_{j \neq i} \bar{Y}_{j\cdot} + \delta)^+] \text{ for } i = 1, 2, ..., k$$

where $-x^- = min[0, x]$ and $x^+ = max[0, x]$ and μ is the mean of solution i.

Results Examination of the MCB confidence intervals provides inferences on the (assumed) superiority of the best design identified above. After having calculated the confidence intervals, one other designs has an MCB interval that contains 0. This means that from an MCB perspective, there is no one uniformly superior design. These 2 designs are all superior to the remaining systems, however, there is no clear 'best' among them as they are all indifferent to each other. Table 4.12 shows the set up of the confidence interval for these two designs that contain 0 in the confidence interval. When presented with these solutions it is tempting to pick the design that yields the lowest goal value (as R&S would). However, one of the benefits of using the NM procedure is that the analyst gains inferences on systems other than the best, which may lead to the selection of an inferior system (i.e. a system with a lower-valued mean that is still within the indifference zone) based on some secondary criteria not reflected in the goal of interest. For instance, now that the designs that are selected that are indifferent to each other, the analyst might be able to use a secondary criteria to make his/her final selection from among the 'best' alternatives. An example of a secondary criteria could be that the analyst might know that there is be a problem with stock delivery for a month in the future and therefore he would chose a design where the stock delivery for that month was the lowest.

Design	$\bar{Y}_{i\cdot}$	lower MCB limit	$\bar{Y}_{i\cdot} - \min_{j \neq i} \bar{Y}_{j\cdot}$	upper MCB limit
1	21.072	-0.274	0.725	1.725
2	20.346	-1.725	-0.725	0.274

Table 4.12: Confidence interval for the 2 best solutions identified by the Nelson-Matejcik method

Table 4.13 shows the average value of each of the objectives. From this table it can be seen that for design 1, the average cost value and the average undersupply value are both larger than that of design 2. Also, the average overstock for design 1 is lower than that for design 2. It can be seen that the difference between each of the objectives for the two designs is rather large and yet, the final goal value that is achieved makes these two solutions indifferent to one another. Design 1 achieves its goal by keeping the average overstock low but by doing this it causes the other two objectives to be somewhat larger. The opposite can be said for design 2 where the goal value is achieved by keeping the average cost value and average undersupply rather low but having the average overstock being rather high.

	Average cost value	Average overstock	Average undersupply
Design 1	1205643	1148	15129
Design 2	1171418	1828	12843
Absolute difference between 1 and 2	34225	680	2286

Table 4.13: Average objective values achieved for selected best designs by the Nelson-Matejcik method

Conclusion

The NM method was applied to the real world stochastic simulation model. From this, the design that yields the lowest goal value is found as well as a design that yields a goal value that is indifferent to that achieved by the design that yields the best goal value. This was done by selecting the values for the maintenance, cost of supply, amount of delivery and quality of the stock that level. Although this method limits itself to performing analysis on only a selection of 1000 designs from the feasible solution space, it is good at providing initial design solutions and goal values. An advantage of this method is its ability to identify solutions that are indifferent to one another (although defining the indifference zone is difficult). This is particularly pertinent to the real world stochastic simulation model as since the goal function is a multi-objective one, there can be more than one way of achieving the minimum goal. Also, since this optimisation is of a simulation model, each time that the simulation model is run the goal value can change. Although this method provided good starting solutions, it did not produce the best solutions compared to the other methods and it takes a very long time to run.

4.3 Comparison on Methods

Now that all the simulation optimisation methods have been found the results are compared to see which method performs the best.

Firstly, a comparison of the methods is done in terms of the time it takes to run each method. These times are rounded to the nearest hour and shown in Table 4.14. It is noted that for the simulated annealing and the genetic algorithm method that decisions are made prior to the method implementation and once the method begins there is no need for any further

decisions to be made by the modeler. In a sense, these two methods are very hands off once the method has been correctly set up. The ordinal optimisation and the Nelson-Matejcik method, however, require input from the modeller throughout the method implementation as certain calculations and decisions need to be made. As Table 4.14 shows, the method that takes the shortest amount to run is simulated annealing, followed by the ordinal optimisation, genetic algorithm and then lastly the Nelson-Matejcik method. It is worth mentioning that most of the time spent when these models run is on the link between the R, the software package used to run the optimisation process, and DIAS, the software package used to run the simulation model. For example, in the simulated annealing method, although the simulation model only takes 46 seconds to run, to populate the input file, upload it into the simulation model, run the simulation model and then gather the output values to calculate the goal takes 1 minute and 20 seconds.

For all methods, the fact that the feasible solution space was so large caused problems in achieving the optimum value. This is due to the fact that there are 700 parameter values that need to be defined in order for the simulation model to run and most of the parameter values have a rather large range. The main parameter that caused difficulty in achieving an optimal solution when applying the simulation optimisation technique was the choice of the stock delivery parameter. This is due to the fact that they make up 168 of the 700 parameters and they each have a very large range over which they are defined.

Method	Time to run (approximate)
Simulated annealing	15 hours
Genetic algorithm	3 days and 5 hours
Ordinal optimisation	1 day and 4 hours
Nelson-Matejcik	3 weeks

Table 4.14: Time to run each simulation optimisation method

Of all the simulation optimisation methods that were implemented, genetic algorithms performed the worst. It was seen that the goal reached a local minimum value and got stuck there and was unable to produce results as good as the other methods did. The other methods managed to produce good solutions. Ordinal optimisation performed well but due to the goal softening approach, the best goal value is not produced but rather a group of solutions that are ranked as the best. The Nelson-Matejcik method performed well and was good at providing solutions that are indifferent to one another but the challenge with this method is that it takes a very long time to run and also specifying the indifference zone. Lastly, simulated annealing produced good results in the shortest amount of time but these solutions were very dependent on the initial solution used.

To furtfor use amount groupsher the research, the simulated annealing and the ordinal optimisation methods were combined to use the strengths that both these methods have to offer. This was done by first running the ordinal optimisation method as defined above. This produced a solution that was ranked as the best in terms of having the lowest goal values across all 1000 possible solutions. Once this solution was generated, it was taken and defined

to be the initial solution when running the simulated annealing algorithm. The simulated annealing algorithm was run as defined above and the result was generated. Through doing this, ordinal optimisation provides a good starting place for simulated annealing algorithm to start and running the simulated annealing algorithm helps to then refine the solution as the whole feasible region can be explored unlike in the ordinal optimisation method.

Figure 4.17 shows the convergence pattern that was generated when running the joint ordinal optimisation and simulated annealing algorithm. The final goal value achieved is 18.316. This goal value was generated on the first implementation of the combined ordinal optimisation and simulated annealing algorithm and provided a very good result on the first implementation of the method.

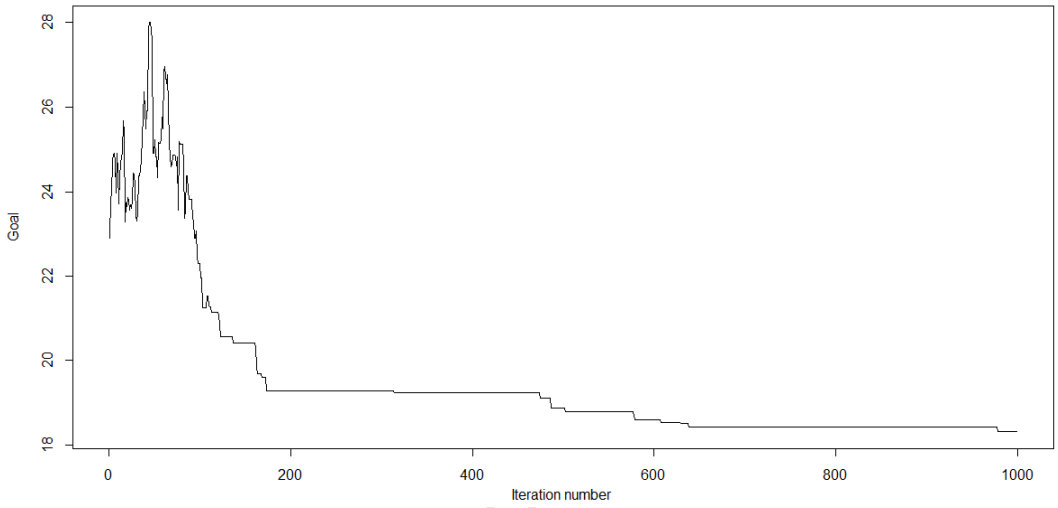

Figure 4.17: Convergence pattern of the goal value for the combined ordinal optimisation and simulated annealing algorithm

Table 4.15 shows the value of the objectives that make up the final goal value for both the simulated annealing method and the joint ordinal optimisation simulated annealing algorithm. The goal value that is achieved for this joint method is only slightly higher than that achieved by the simulated annealing method (which can be attributed to simulation noise) but it has the advantages that it was generated on the first implementation of the method unlike that of the goal generated by the simulated annealing method alone. The goal for the simulated annealing algorithm is the best out of 30 runs of the simulated annealing simulation optimisation technique. When comparing the objectives that make up the goal for each method, it can be seen that the cost value is lower for the joint method than for the simulated annealing method but that the joint method has a slightly higher overstock and undersupply value.

Method	Goal value	Cost value	Overstock value	Undersupply value
Joint ordinal optimisation and simulated annealing	18.316	1126909	2130	6369
Simulated annealing	18.278	1152559	2083	5179

Table 4.15: Final solution from joint ordinal optimisation and simulated annealing method

Chapter 5. Conclusion

5.1 Conclusion

The aim of this thesis that was met was to find a simulation optimisation technique which was best suited to optimise a real world simulation model which was provided by a client in the energy sector.

In order to achieve this aim, an understanding of the possible simulation optimisation techniques available was required. This was done by conducting an extensive literature review which not only looked at how simulation optimisation was defined but also at the simulation optimisation methods available.

It was seen that simulation optimisation can be formally defined by

$$\min_{\theta \in \Theta} J(\theta) = E[L(\theta, \omega)]$$

where $\theta \in \Theta$ represents the (vector of) input variables, $J(\theta)$ is the objective function, ω represents the sample path (simulation replication) and L is the sample performance measure. Due to the fact that the performance of $J(\theta)$ cannot be evaluated analytically the expectation of the performance L is taken. The constraint set Θ may be either explicitly given or implicitly defined [27].

There are many available simulation optimisation techniques and as such they were grouped, based on their properties, into four main categories defined as: heuristic solution methods, gradient based search methods, statistical methods and response surface methodology. These methods were explained and the various advantages and disadvantages of each of the methods were given.

Before any simulation optimisation techniques were applied to the real life simulation model provided by the client, a selection of methods was first applied to a test case simulation model (which was created in house) to get a better understanding of how these methods work. The test case simulation model contains inherent randomness and is a very much simplified model that models the functioning of Open Cycle Gas Turbines. The model can be described such that it considers policy decisions of fuel orders based on forecasted demands done months into the future, after which actual demands are simulated to generate changes in stock, fuel shortages and unnecessary burning of excess fuels. The policies considered by the model relate decisions made (fuel orders) to parameters of the probability distribution on forecasted demands. Throughout

the running of the model, the costs that are incurred are measured. The aim was then to optimise the model by minimising the goal where the goal was defined as incurring as little cost as possible.

Five simulation optimisation techniques were selected to be applied to this test case simulation model based on the fact that they appeared to be best suited to the problem at hand. These were Genetic Algorithm, Simulated Annealing, Nested Partitions, Nelson-Matejcik method and Ordinal Optimsation. As expected, each of these methods had their own strengths as to why they suited the simulation optimisation problem at hand. These are shown in Table 5.1. Of the methods discussed in the literature review, these seem to be the best suited methods to the simulation model at hand. These methods were then implemented to optimise the test case simulation model and through this implementation a better understanding of these methods is gathered. It is noted that this is very much a test case model and therefore the results gathered can in no way be used in a real life setting.

When optimising this test case simulation model it was seen that there were many parameter values that were able to achieve the optimum value and the methods used were good at identifying this. Of all the methods that were implemented, the best solutions were achieved by the Genetic Algorithm, followed by the Simulated Annealing algorithm. The Nested Partitions method performed well in a short amount of time. The Nelson and Matejcik method did not produce optimum answers as well as the Ordinal Optimisation method.

Through this knowledge and understanding, the problem brought by the client could be tackled. The client has a real world simulation model, which models the production of 14 of the plants that they own. The simulation model was created by the client and brought to be optimised so that the simulation model could be used not only to provide a what-if analysis but also provide information, once optimised, as to what the various input values would create the best output based on predefined constraints. In order to achieve this it meant that the client wanted the feature of optimisation added to their current simulation model.

Before any of the five simulation optimisation techniques discussed above could be implemented, the simulation model itself had to be understood in terms of the input and output values that were used and generated by the model. Also, the definitions of what exactly the client wanted to optimise needed to be defined. In order to do this a workshop was held with the client. The workshop started out by giving a brief overview of the simulation model, the inputs and outputs that the model has, as well as the internal structure of the model. An explanation as to the key concepts underlying the search for 'optimal' policies using simulation models was given. After this, the nominal group technique was used to illicit the answer to the question of what the key policy objectives to be optimised are. From this the ideas that were established were clustered and structured into policy actions and objectives over both a short and long term horizon. This important information was then used to define the objectives which the simulation optimisation method should minimise and since the problem is dealing with the short term only the objectives defined over the short terms would be focused on.

Once this was established, a goal value was generated which was a multi-objective value that was made up of three identified objectives. The goal value is a quantity that is made up of the

Method	Strengths
Genetic Algorithms	Produce good results Well known method Easy to implement and are adaptable to suit the problem at hand as well as being easy to modify Able to tolerate noisy, discontinuous and time varying functions and only need to be able to calculate some measure of performance [85] Allow for a number of different implementation choices [85] Inherently robust [85] Use the accumulative information about the initial unknown search space in order to move the next searches into useful spaces [8] Do not appear to need a good set on initial solutions to reach a good solution [8]
Simulated Annealing	Produce good results Well known method Easy to implement Able to tolerate noisy, discontinuous and time varying functions Does not require any fundamental derivative information [34]
Nested Partitions	Interesting and new method Does not need a good starting solution to be implemented Spends most computational effort in the region that is considered the most promising at any given iteration [77] Handles noise objective functions Few assumptions are made about the structure of the problem [96] Fast method to implement
Ordinal Optimisaiton	Easy to implement and applicable to a large class of problems Focuses on order rather than value Practical method as it fulfills the engineering requirements for a useful and satisfying solution without any extreme claims on the degree of the optimality of the solution [114] Estimation and comparison can be done based on very short observations or very approximate models and therefore considerable simulation work can be reduced [50]
Nelson-Matejcik Method	Combines the two statistical methods of multiple comparison with the best and ranking and selection procedures and therefore incorporating the advantages of these two methods Well suited to the problems with noisy objective functions Allow secondary criterion not reflected in the performance measure to be included [72]

Table 5.1: Strengths of the five simulation optimisation techniques selected from the literature review

various objectives identified by the client and its value is used by the simulation optimisation technique when trying to find the 'optimal' solution. The aim of the optimisation techniques is to minimise this goal value. The three objectives used to create the goal value were to minimise the cost of production, minimise the total amount of overstock and minimise the total amount of undersupply. These three objective values were combined and the resulting goal value was defined as:

$$\text{minimise} \left\{ \max_{i=1,2,3} \frac{(z_i - I_i)}{q_i} + \epsilon \sum_{i=1}^{3} w_i z_i \right\}$$

where ϵ is equal to 0.1, I_i is the ideal value of objective i, z_i is the value of objective i, q_i is the range of possible values for that objective obtained from the payoff table (a more detailed explanation of this table is given in the chapter on the real life simulation model) and w_i is the weight associated with objective i.

The simulation optimisation techniques were then implemented. The five simulation optimisation techniques that were implemented on the test case simulation model were then applied to the real world simulation model. It was seen, however, that the Nested Partitions method could not be implemented due to the fact that the size of the problem (in terms of number of parameters) at hand was too large for the optimisation technique to deal with.

The advantages and disadvantages of these methods are listed in Table 5.2. Of these methods, the two that seemed to be the most promising were that of Simulated Annealing and Ordinal Optimisation. As a result these two methods were combined and it was found that it was this model that produced the 'best' results in terms of minimising the goal. The reason as to why 'best' is in inverted commas is due to the fact that there is randomness in the model and therefore no way to guarantee that the solution found is the optimal solution.

Overall, it is recommended that the simulation optimisation method that should be used when dealing with the optimisation of this real life simulation model is a combination of the Ordinal Optimisation algorithm and the Simulated Annealing algorithm. It is worth noting that due to the inherent randomness present in the simulation optimisation problem there is no way to guarantee that the best solution will be found. It was however seen that the use of the combined Ordinal Optimisation and Simulated Annealing method provided good results and therefore is a good technique to use.

Through this research, the goal of providing an optimisation technique to optimise a simulation model has been achieved. The client will now be able to use their simulation model not only for 'what-if' analysis but also add the feature of optimisation to it which will ideally help in their being able to best achieve their objectives.

5.2 Future work

Although recommendations have been made in terms of the most suitable simulation optimisation technique to be implemented on the real world simulation model, this is not to say that there is no further research to be made.

Firstly, only five out of many possible simulation optimisation techniques were chosen to

Simulation Optimisation Method	Simulated Annealing	Genetic Algorithm	Ordinal Optimisation	Nelson-Matejcik Method
Features	Method that is based on the thermodynamic cooling of molten metals to a steady-state minimum value	A method that is modelled on the natural evolution process to find the most 'elite' solution	A method that applies a goal softening approach to the problem to get the best ranked design	A method which combines ranking and selection and multiple comparison with the best
Advantages	-Provides best optimal solutions provided that the initial model is well selected This method is fast to run	-Makes use of the features of natural selection allowing the good features of the parameters to move from one generation to the next -The feasible solution space is well searched due to the presence of a mutation rate and many children solutions	Uses a crude model which allows computational time to decrease and as such the method is fast to implement No prior knowledge of the simulation model is required Having a goal softening approach when dealing with simulation model optimisation is attractive as goal value can change from one simulation run to the next Provides a group of solutions that lie withing the to x percentage of optimal solutions	Can calculate many solutions within the indifference zone therefore providing a group of possible solutions and not one optimum solution No prior knowledge of the simulation model is required Through the running the same designs through the simulation model many times variance reduction takes place Use of common random numbers decreases noise achieved by simulation model
Disadvantages	Results are very dependent on the initial model parameter values Method has to be tweaked to suit the simulation model that is dealt with	Difficulty in handling the large search space The binary bit encoding of the continuous parameters limits the quality of the solution Requires that the method is tweaked to suit the simulation model Method takes a long time to run	Can be difficult to come up with a crude model that represents system well The method only deals with a selection of solutions from the feasible space and as such the quality of the solution is limitted	Takes a very long time to run Difficulty in coming up with an indifference zone when dealing with a multi-objective goal This method only deals with a selection of solutions from the feasible space and as such the quality of the solution is limitted

Table 5.2: Advantages and disadvantages of the simulation optimisation methods

be implemented on the simulation models. The field of simulation optimisation is growing at a rapid rate and therefore there could be new methods that are more suitable to optimise a system than those that were tried in this thesis.

There were certain challenges that were faced when applying the optimisation techniques; some of which were linked to the simulation optimisation process and others were side complications. Although many of these were dealt with in this thesis, future research could go into looking at these in more detail. A list of these are given below.

- An optimisation model is only able to optimise the goal that you give it. This means that even if the optimisation model works extremely well, if the goal value is not defined in a way that captures all the desired objectives then the result achieved might not be correct. As such, further research could be spent on trying to better define the goal that the client would want to optimise in terms of the objectives that it includes as well as the various importance weights of the objectives. More specifically, further work can be done with the client when defining the objective of cost of production to move from measuring the cost of production on an ordinal scale to a cardinal scale.

- In the real life simulation model, the size of the possible solution space is very large and, as such, caused problems with some of the optimisation techniques, specifically Genetic Algorithm. Future research could go into trying to refine the search space by tightening the constraints of some of the parameter values, particularly for the parameter dealing with the delivery of stock. This would then lead to a smaller solution space which would make it easier for the optimum goal value to be found.

- Future work on each of the simulation optimisation techniques themselves. These are defined for each method as follows:

  - *Simulated Annealing* - For this method, it would be worthwhile looking into increasing the running time of the simulation optimisation algorithm to see if this could lead to better results. Also, research can be done in terms of the initial temperature value that is used when performing the algorithm.
  - *Genetic Algorithm* - From the results that were generated it was seen that the Genetic Algorithm optimisation technique did not perform well when trying to optimise the real life simulation model. Future research could be spent on trying to see if the relevant preliminary information could be selected in such a way that would make this method a successful one. This is specifically with reference to the generating of children and the application of the mutation rate.
  - *Ordinal Optimisation* - Further research can be done in defining the crude model to be used as well as potentially increasing the number of solutions selected from the solution space to be tested.
  - *Nelson-Matejcik method* - Although there is potential benefit in looking into better defining the indifference zone used in the method, the results generated indicated

that this method was not a successful one when trying to optimise this simulation model and therefore further research work would be more beneficial for the methods that appear most promising.

Bibliography

[1] E.H.L. Aarts and P.J.M. van Laarhoven. Simulated annealing: An introduction. *Statistica Neerlandica*, 43(1):31–52, 1989.

[2] M.A. Ahmed and T.M. Alkhamis. Simulation-based optimization using simulated annealing with ranking and selection. *Computers and Operations Research*, 29(4):387–402, 2002.

[3] C. Alexopoulos and A.F. Seila. Implementing the batch means method in simulation experiments. In *Proceedings of the 28th Winter Simulation Conference*, WSC '96, pages 214–221, Washington, DC, USA, 1996. IEEE Computer Society.

[4] M.H. Alrefaei and S. Andradottir. A simulated annealing algorithm with constant temperature for discrete stochastic optimization. *Management Science*, 45(5):748–764, 1999.

[5] S. Andradóttir. A review of simulation optimization techniques. In *Proceedings of the 30th Winter Simulation Conference*, WSC '98, pages 151–158, Los Alamitos, CA, USA, 1998. IEEE Computer Society.

[6] S. Andradottir. Chapter 20: An overview of simulation optimization via random search. In S. G. Henderson and B. L. Nelson, editors, *Simulation*, volume 13 of *Handbooks in Operations Research and Management Science*, pages 617–631. Elsevier, 2006.

[7] H.M. Antia. *Numerical Methods for Scientists and Engineers*. Birkhäuser Basel, 2002.

[8] A. Azadeh and S. Tarverdian. Integration of genetic algorithm, computer simulation and design of experiments for forecasting electrical energy consumption. *Energy Policy*, 35(10):5229–5241, 2007.

[9] F. Azadivar. A tutorial on simulation optimization. In *Proceedings of the 24th Winter Simulation Conference*, WSC '92, pages 198–204, New York, NY, USA, 1992. ACM.

[10] F. Azadivar. Simulation optimization methodologies. In *Proceedings of the 31stWinter Simulation Conference: Simulation—a bridge to the future - Volume 1*, WSC '99, pages 93–100, New York, NY, USA, 1999. ACM.

[11] J. Banks. *Discrete-Event System Simulation, Fifth Edition*. Prentice Hall, 2010.

[12] B. Baran, E. Kaszkurewicz, and A. Bhaya. Parallel asynchronous team algorithms: Convergence and performance analysis. *IEEE Transactions on Parallel and Distributed Systems*, 7(7):677–688, 1996.

[13] R.R. Barton and J.S. Ivey. Nelder-mead simplex modifications for simulation optimization. *Management Science*, 42(7):954–973, 1996.

[14] M. Better, F. Glover, G. A. Kochenberger, and H. Wang. Simulation optimization: Applications in risk management. *International Journal of Information Technology and Decision Making*, 7(4):571–587, 2008.

[15] C. Blum. Ant colony optimization: Introduction and recent trends. *Physics of Life Reviews*, 2(4):353–373, 2005.

[16] J. Boesel, B.L. Nelson, and S. Kim. Using ranking and selection to 'clean up' after simulation optimization. *Operations Research*, 51(5):814–825, 2003.

[17] X. Cao. Perturbation analysis of discrete event systems: Concepts, algorithms, and applications. *European Journal of Operational Research*, 91(1):1–13, 1996.

[18] Y. Carson and A. Maria. Simulation optimization: Methods and applications. In *Proceedings of the 29th Winter Simulation Conference*, WSC '97, pages 118–126, Washington, DC, USA, 1997. IEEE Computer Society.

[19] A. Cave, S. Nahavandi, and A. Kouzani. Schedule evaluation: Simulation optimization for process scheduling through simulated annealing. In *Proceedings of the 34th Winter Simulation Conference: Exploring New Frontiers*, WSC '02, pages 1909–1913. Winter Simulation Conference, 2002.

[20] G.M. Clark and W. Yang. A bonferroni selection procedure when using commom random numbers with unknown variances. In *Proceedings of the 18th Winter Simulation Conference*, WSC '86, pages 313–315, New York, NY, USA, 1986. ACM.

[21] H.A. David and H.N. Nagaraja. *Order Statistics*. Wiley series in probability and mathematical statistics. Probability and mathematical statistics. Wiley, 2004.

[22] B. Dengiz and C. Alabas. Simulation optimization using tabu search. In *Simulation Conference, 2000. Proceedings. Winter*, volume 1, pages 805–810, 2000.

[23] E.J. Dudewicz and S.R. Dalal. Allocation of observations in ranking and selection with unequal variances. *Sankhya-: The Indian Journal of Statistics, Series B (1960-2002)*, 37(1):28–78, 1975.

[24] R.W. Eglese. Simulated annealing: A tool for operational research. *European Journal of Operational Research*, 46(3):271–281, 1990.

[25] S. S. Fan and E. Zahara. A hybrid simplex search and particle swarm optimization for unconstrained optimization. *European Journal of Operational Research*, 181(2):527–548, 2007.

[26] D. Fouskakis and D. Draper. Stochastic optimization: a review. *International Statistical Review*, 70(3):315–349, 2002.

[27] M. C. Fu. Optimization for simulation: Theory vs.practice. *INFORMS Journal on Computing*, 14(3):192, 2002.

[28] M. C. Fu, S. Andradóttir, J. S. Carson, F. Glover, C. R. Harrell, Y. Ho, J. P. Kelly, and S. M. Robinson. Integrating optimization and simulation: Research and practice. In *Proceedings of the 32nd Winter Simulation Conference*, WSC '00, pages 610–616, San Diego, CA, USA, 2000. Society for Computer Simulation International.

[29] M. C. Fu, F. W. Glover, and J. April. Simulation optimization: a review, new developments, and applications. In *Proceedings of the 37th Winter Simulation Conference*, WSC '05, pages 83–95. Winter Simulation Conference, 2005.

[30] M.C. Fu. A tutorial review of techniques for simulation optimization. In *Winter Simulation Conference Proceedings, 1994.*, pages 149–156, 1994.

[31] M.C. Fu. Simulation optimization. In *In Proceeding of the 2001 Winter Simulation Conference*, pages 53–61. Piscataway, 2001.

[32] M.C. Fu. Chapter 19: Gradient estimation. In S. Henderson and B. Nelson, editors, *Simulation*, volume 13 of *Handbooks in Operations Research and Management Science*, pages 575–616. Elsevier, 2006.

[33] M.C. Fu, C. Chen, and L. Shi. Some topics for simulation optimization. In *Proceedings of the 40th Winter Simulation Conference*, WSC '08, pages 27–38. Winter Simulation Conference, 2008.

[34] T. Gál, T.J. Stewart, and T. Hanne. *Multicriteria Decision Making: Advances in McDm Models, Algorithms, Theory, and Applications*. International Series in Operations Research & Management Science, 21. Springer-Verlag GmbH, 1999.

[35] F. Glover, J.P. Kelly, and M. Laguna. New advances and applications of combining simulation and optimization. In *Proceedings of the 28th Winter Simulation Conference*, WSC '96, pages 144–152, Washington, DC, USA, 1996. IEEE Computer Society.

[36] F. Glover, M. Laguna, and R. Marti. Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 39:653–684, 2000.

[37] F.W. Glover and F.G.M. Laguna. *Tabu Search*. Number v. 1 in Tabu Search. Springer-Verlag GmbH, 1998.

[38] P. W. Glynn. Likelihood ratio derivative estimators for stochastic systems. In *Proceedings of the 21st Winter Simulation Conference*, WSC '89, pages 374–380, New York, NY, USA, 1989. ACM.

[39] P.W. Glynn. Likelihood ratio gradient estimation: An overview. In *Proceedings of the 19th Winter Simulation Conference*, WSC '87, pages 366–375, New York, NY, USA, 1987. ACM.

[40] D. Goldsman and B.L. Nelson. Statistical screening, selection, and multiple comparison procedures in computer simulation. In *Proceedings of the 30th Winter Simulation Conference*, WSC '98, pages 159–166, Los Alamitos, CA, USA, 1998. IEEE Computer Society.

[41] H. Gonda Neddermeijer, G.J. van Oortmarssen, N. Piersma, and R. Dekker. A framework for response surface methodology for simulation optimization. *Winter Simulation Conference*, 1:129–136, 2000.

[42] A. G. Greenwood, L. Paul Rees, and F. C. Siochi. An investigation of the behavior of simulation response surfaces. *European Journal of Operational Research*, 110(2):282–313, 1998. EURO Best Applied Paper Competition.

[43] X. Guan, Y. Ho, and F. Lai. An ordinal optimization based bidding strategy for electric power suppliers in the daily energy market. *IEEE Transactions on Power Systems*, 16(4):788 –797, nov 2001.

[44] W. Hachicha, A. Ammeri, F. Masmoudi, and H. Chachoub. A comprehensive literature classification of simulation optimisation methods. MPRA Paper 27652, University Library of Munich, Germany, 2010.

[45] R.G. Haight and L.E. Travis. Wildlife conservation planning using stochastic optimization and importance sampling. *Forest Science*, 43(1):129–139, 1997.

[46] B. Hajek. Cooling schedules for optimal annealing. *Mathematics of Operations Research*, 13(2):311–329, 1988.

[47] R.L. Haupt and S.E. Haupt. *Practical Genetic Algorithms*. Wiley-Interscience publication. Wiley, 2004.

[48] D. Henderson, S. Jacobson, and A. Johnson. The theory and practice of simulated annealing. In Fred Glover and Gary Kochenberger, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, pages 287–319. Springer New York, 2003.

[49] Y. Ho. Overview of ordinal optimization. In *Proceedings of the 33rd IEEE Conference on Decision Control*, volume 2, pages 1975–1977, 1994.

[50] Y. Ho, R. Sreenivas, and P. Vakili. Ordinal optimization of deds. *Discrete Event Dynamic Systems*, 2:61–88, 1992.

[51] Y.C. Ho, Q.C. Zhao, and Q.S. Jia. *Ordinal Optimization: Soft Optimization for Hard Problems*. The International Series on Discrete Event Dynamic Systems. Springer, 2007.

[52] Y. Hochberg and A.C. Tamhane. *Multiple Comparison Procedures*. Wiley Series in Probability and Statistics. Wiley, 2009.

[53] S.J. Hood and P.D. Welch. Response surface methodology and its application in simulation. In *Proceedings of the 25th Winter Simulation Conference*, WSC '93, pages 115–122, New York, NY, USA, 1993. ACM.

[54] B. Hsieh, C. Chen, and S. Chang. Scheduling semiconductor wafer fabrication by using ordinal optimization-based simulation. *IEEE Transactions on Robotics and Automation*, 17(5):599–608, 2001.

[55] J.C. Hsu. Constrained simultaneous confidence intervals for multiple comparisons with the best. *The Annals of Statistics*, 12(3):1136–1144, 1984.

[56] R.A. Jabr and B.C. Pal. Ordinal optimisation approach for locating and sizing of distributed generation. *Generation, Transmission Distribution, IET*, 3(8):713–723, august 2009.

[57] Y. Jaluria. Simulation-based optimization of thermal systems. *Applied Thermal Engineering*, 29(7):1346–1355, 2009. Selected Papers from the 10th UK National Heat Transfer Conference, Edinburgh, Scotland, September 10-11, 2007.

[58] Q. Jia, Y. Ho, and Q. Zhao. Comparison of selection rules for ordinal optimization. *Mathematical and Computer Modelling*, 43(910):1150–1171, 2006.

[59] S. Juneja. Rare-event simulation techniques: An introduction and recent advances. In *Handbook of Simulation, volume 13 of Handbooks in Operations Research and Management Science*, pages 291–350. Elsevier, 2006.

[60] J. Kiefer and J. Wolfowitz. Stochastic estimation of the maximum of a regression function. *The Annals of Mathematical Statistics*, 23(3):462–466, 1952.

[61] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[62] J.P.C. Kleijnen. Response surface methodology for constrained simulation optimization: An overview. *Simulation Modelling Practice and Theory*, 16(1):50–64, 2008.

[63] A. Konak and S. Kulturel-Konak. Simulation optimization using tabu search: An emperical study. In *Proceedings of the 37th Winter Simulation Conference*, WSC '05, pages 2686–2692. Winter Simulation Conference, 2005.

[64] T. Lacksonen. Empirical comparison of search algorithms for discrete event simulation. *Computers and Industrial Engineering*, 40(1-2):133–148, 2001.

[65] T.W.E. Lau and Y.C. Ho. Universal alignment probabilities and subset selection for ordinal optimization. *Journal of Optimization Theory and Applications*, 93:455–489, 1997.

[66] A.M. Law and W.D. Kelton. *Simulation modeling and analysis*. McGraw-Hill series in industrial engineering and management science. McGraw-Hill, 2000.

[67] P. L'Ecuyer. A unified view of the ipa, sf, and lr gradient estimation techniques. *Management Science*, 36(11):1364–1383, 1990.

[68] P. L'Ecuyer, N. Giroux, and P. W. Glynn. Stochastic optimization by simulation: Numerical experiments with the m/m/1 queue in steady-state. *Management Science*, 40(10):1245–1261, 1994.

[69] A. Maria. Introduction to modeling and simulation. In *Proceedings of the 29th Winter Simulation Conference*, WSC '97, pages 7–13, Washington, DC, USA, 1997. IEEE Computer Society.

[70] M. Marseguerra and E. Zio. Optimizing maintenance and repair policies via a combination of genetic algorithms and monte carlo simulation. *Reliability Engineering and; System Safety*, 68(1):69–83, 2000.

[71] R. Marti, M. Laguna, and F. Glover. Principles of scatter search. *European Journal of Operational Research*, 169(2):359–372, 2006. Feature Cluster on Scatter Search Methods for Optimization.

[72] F.J. Matejcik and B.L. Nelson. Simultaneous ranking, selection and multiple comparisons for simulation. In *Proceedings of the 25th Winter Simulation Conference*, WSC '93, pages 386–392, New York, NY, USA, 1993. ACM.

[73] K. Miettinen, J. Branke, K. Deb, and R. Slowinski. *Multiobjective Optimization: Interactive and Evolutionary Approaches*. Lecture Notes in Computer Science / Theoretical Computer Science and General Issues. Springer, 2008.

[74] M. Mitra and S.K. Park. Solution to the indexing problem of frequency domain simulation experiments. In *Proceedings of the 23rd Winter Simulation Conference*, WSC '91, pages 907–915, Washington, DC, USA, 1991. IEEE Computer Society.

[75] H.G. Neddermeijer. *Adaptive Extensions of the Nelder and Mead Simplex Method for Optimization of Stochastic Simulation Models*. Econometric Institute, 2000.

[76] B.L. Nelson and F.J. Matejcik. Using common random numbers for indifference-zone selection and multiple comparisons in simulation. *Management Science*, 41(12):1935–1945, 1995.

[77] S. Ólafsson. Chapter 21: Metaheuristics. In S. G. Henderson and B. L. Nelson, editors, *Simulation*, volume 13 of *Handbooks in Operations Research and Management Science*, pages 633–654. Elsevier, 2006.

[78] I.H. Osman. Preface: Focused issue on applied meta-heuristics. *Computers and Industrial Engineering*, 44(2):205–207, 2003.

[79] B. Ozcelik and T. Erzurumlu. Determination of effecting dimensional parameters on warpage of thin shell plastic parts using integrated response surface method and genetic algorithm. *International Communications in Heat and Mass Transfer*, 32(8):1085–1094, 2005.

[80] K. Parhi and R. Berkowitz. On optimizing importance sampling simulations. *IEEE Transactions on Circuits and Systems*, 34(12):1558–1563, 1987.

[81] R. J. Paul and T. S. Chanev. Simulation optimisation using a genetic algorithm. *Simulation Practice and Theory*, 6(6):601–611, 1998.

[82] C.D. Pegden and M.P. Gately. Decision optimization for gasp iv simulation models. In *Proceedings of the 9th Winter simulation Conference - Volume 1*, WSC '77, pages 126–133. Winter Simulation Conference, 1977.

[83] M. Pirlot. General local search methods. *European Journal of Operational Research*, 92(3):493–511, 1996.

[84] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2012. ISBN 3-900051-07-0.

[85] C. R. Reeves. Genetic algorithms for the operations researcher. *INFORMS Journal on Computing*, 9(3):231–250, 1997.

[86] Y. Rinott. On two-stage selection procedures and related probability-inequalities. *Communications in Statistics - Theory and Methods*, 7(8):799–811, 1978.

[87] H. Robbins and S. Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.

[88] S. Robinson. Discrete-event simulation: From the pioneers to the present, what next? *The Journal of the Operational Research Society*, 56(6):619–629, 2005.

[89] D. Ruppert, R.L. Reish, R.B. Deriso, and R.J. Carroll. Optimization using stochastic approximation and monte carlo simulation (with application to harvesting of atlantic menhaden). *Biometrics*, 40(2):535–545, 1984.

[90] M. Safizadeh and B.M. Thornton. Optimization in simulation experiments using response surface methodology. *Computers and Industrial Engineering*, 8(1):11–27, 1984.

[91] B. Schmeiser. Batch size effects in the analysis of simulation output. *Operations Research*, 30(3):556–568, 1982.

[92] L.W. Schruben and V.J. Cogliano. Simulation sensitivity analysis: A frequency domain approach. In *Proceedings of the 13th Winter Simulation Conference - Volume 2*, WSC '81, pages 455–459, Piscataway, NJ, USA, 1981. IEEE Computer Society.

[93] L.W. Schruben and V.J. Cogliano. An experimental procedure for simulation response surface model identification. *Communications of the ACM*, 30(8):716–730, 1987.

[94] L. Shi. A new algorithm for stochastic discrete resource allocation optimization. *Discrete Event Dynamic Systems*, 10:271–294, 2000.

[95] L. Shi, C. Chen, and E. Yucesan. Simultaneous simulation experiments and nested partition for discrete resource allocation in supply chain management. In *Winter Simulation Conference Proceedings, 1999*, volume 1, pages 395–401, 1999.

[96] L. Shi and S. Ólafsson. Nested partitions method for global optimization. *Operations Research*, 48(3):390–407, 2000.

[97] L. Shi and S. Ólafsson. *Nested Partitions Method, Theory and Applications*. International Series in Operations Research & Management Science. Springer, 2008.

[98] E. A. Silver. An overview of heuristic solution methods. *The Journal of the Operational Research Society*, 55(9):936–956, 2004.

[99] D. Sink. Using the nominal group technique effectively. *National Productivity Review*, 2(2):173–184, 1983.

[100] S.N. Sivanandam and S.N. Deepa. *Introduction to Genetic Algorithms*. Springer-Verlag Berlin Heidelberg, 2007.

[101] P.J. Smith, M. Shafi, and H. Gao. Quick simulation: a review of importance sampling techniques in communications systems. *IEEE Journal on Selected Areas in Communications*, 15(4):597–613, 1997.

[102] J.C. Spall. *Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control*. Wiley Series in Discrete Mathematics and Optimization. John Wiley & Sons, 2005.

[103] M. Srinivas and L.M. Patnaik. Genetic algorithms: A survey. *Computer*, 27(6):17–26, 1994.

[104] B. Suman. Study of simulated annealing based algorithms for multi-objective optimization of a constrained problem. *Computers and Chemical Engineering*, 28(9):1849–1871, 2004.

[105] J. R. Swisher, S. H. Jacobson, and E. Yücesan. Discrete-event simulation optimization using ranking, selection, and multiple comparison procedures: A survey. *ACM Transactions on Modeling and Computer Simulation*, 13(2):134–154, 2003.

[106] J.R. Swisher and S.H. Jacobson. Evaluating the design of a family practice healthcare clinic using discrete-event simulation. *Health Care Management Science*, 5:75–88, 2002.

[107] S. Talukdar, S. Murthy, and R. Akkiraju. Asynchronous teams. In Fred Glover and Gary Kochenberger, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research and Management Science*, pages 537–556. Springer New York, 2003.

[108] E. Tekin and I. Sabuncuoglu. Simulation optimization: A comprehensive review on theory and applications. *IIE Transactions*, 36(11):1067–1081, 2004.

[109] J. Thompson and K. Dowsland. General cooling schedules for a simulated annealing based timetabling system. In Edmund Burke and Peter Ross, editors, *Practice and Theory of Automated Timetabling*, volume 1153 of *Lecture Notes in Computer Science*, pages 345–363. Springer Berlin / Heidelberg, 1996.

[110] P.J. van Laarhoven and E.H. Aarts. *Simulated Annealing: Theory and Applications*. Mathematics and Its Applications. Springer, 1987.

[111] J.M. Varanelli and J.P. Cohoon. A fast method for generalized starting temperature determination in homogeneous two-stage simulated annealing systems. *Computers and Operations Research*, 26(5):481–503, 1999.

[112] K. O. Willis and D. F. Jones. Multi-objective simulation optimization through search heuristics and relational database analysis. *Decision Support Systems*, 46(1):277–286, 2008.

[113] M. Xie, J. Zhong, and F.F. Wu. Multiyear transmission expansion planning using ordinal optimization. *IEEE Transactions on Power Systems*, 22(4):1420–1428, nov. 2007.

[114] X. Xie. Dynamics and convergence rate of ordinal comparison of stochastic discrete-event systems. *IEEE Transactions on Automatic Control*, 42(4):586–590, 1997.

[115] H. Youssef, S. M. Sait, and H. Adiche. Evolutionary algorithms, simulated annealing and tabu search: A comparative study. *Engineering Applications of Artificial Intelligence*, 14(2):167–181, 2001.

[116] E. Yucesan, Y. Luo, C. Chen, and I. Lee. Distributed web-based simulation experiments for optimization. *Simulation Practice and Theory*, 9(1-2):73–90, 2001.