Differential Evolution Optimisation of a Two Echelon Inventory System

Fergus Haverty

M.Sc. in Computer Science

NUI MAYNOOTH

Ollscoil na hÉireann Má Nuad

Department of Computer Science

National University of Ireland, Maynooth

County Kildare, Ireland

Head of Department: Dr Adam Winstanley

Supervisor: Prof. Ronan Reilly

January 30th, 2011

Contents

Abstract:

In this thesis, we consider the effectiveness of Differential Evolution as a computational algorithm in the context of spare part inventory optimisation in a two-echelon supply chain setting. The study of DE in application to supply chain planning is limited; thus, we present a well-known hard discrete non-linear stochastic optimisation problem to determine the inventory investment for a two-echelon supply chain system. The underlying optimisation problem is notoriously difficult since it has discrete variables and is combinatorial in nature. We assume the best-in-class algorithm in the field of Operations Research to be the optimum optimisation algorithm for the problem under study. We aim to determine the accuracy of DE in generating to the best-in-class optimum solutions for the problem. We select DE control parameters that best fit the problem and select two DE Algorithm variants for the analysis. We create a series of complex conditions in which we expect DE to achieve the optimum solutions and present our case for comparison to the best-in-class method. We show specific enhancements to the DE algorithm to provide a robust method of achieving the optimum solutions unique to the problem under study. In addition, we derive conclusions on the scalability of DE and which DE algorithm is the preferred algorithm for the problem under study. Further, we formulate conclusions under which problem conditions; each DE Algorithm is best suited to achieve the optimum solutions and which DE Algorithm is comparable to the best-in-class analytical method. Finally, we present areas of further research for the application of DE in the context of inventory supply chain.

List of Figures

List of Tables

Acknowledgements

This thesis is the outcome of invaluable contributions from a number of people that I have benefited from over the course of this study. First and foremost, I would like to pay thanks to my supervisor, prof. Ronan Reilly who has supported me over the last year with his wisdom and advice on Differential Evolution and the insight and motivating discussions we have on general Computer Science topics every so often. I would also like to pay thanks to two colleagues from Bell Labs who have provided invaluable mentoring support to me. Dr Mustafa Dogru and Dr Ulas Ozen are researchers in the field of Operations Research with extensive expertise in the field of multi-echelon supply chain systems. Without their support and encouragement to develop the problem domain and mathematical concept, I would not have reached this point. Finally, I would like to thank my new wife Suzanne, as we got married during the course of this study. Many a late night was lost in the development of this thesis and I thank Suzanne so much for listening to me and understanding.

Chapter 1

1 Introduction

Differential Evolution (DE) is a powerful, easy to use, fast, reliable global optimisation algorithm for tackling difficult optimisation problems. It is ideal for researchers who need a robust optimisation algorithm without the need for expertise in the field of evolutionary algorithms. In this thesis, we use DE to optimise a common model in the domain of supply chain planning and distribution. The model is a multi-echelon system comprising of one central warehouse and multiple local warehouses. See Figure 1.1. It is also more commonly referred to in the literature as a one-warehouse multi-retailer system. The system is employed across a wide range of domains from retail to manufacturing. It is considered a complex optimisation problem as it contains a number of parameters that vary the complexity of the model under certain conditions. The goal is to apply DE as an optimisation algorithm to match the solutions of the best-in-class analytical algorithm in the field of Operations Research.

1.1 A Two-Echelon Inventory System for Spare Parts

In relation to the telecommunications sector, Internet Service Providers (ISPs) must maintain their network to avoid customer attrition. Any outages can negatively impact consumer sentiment. Similar to the reconfiguration of manufacturing processes into a supply chain paradigm, companies are looking at the services business as a new way to differentiate themselves from competitors.

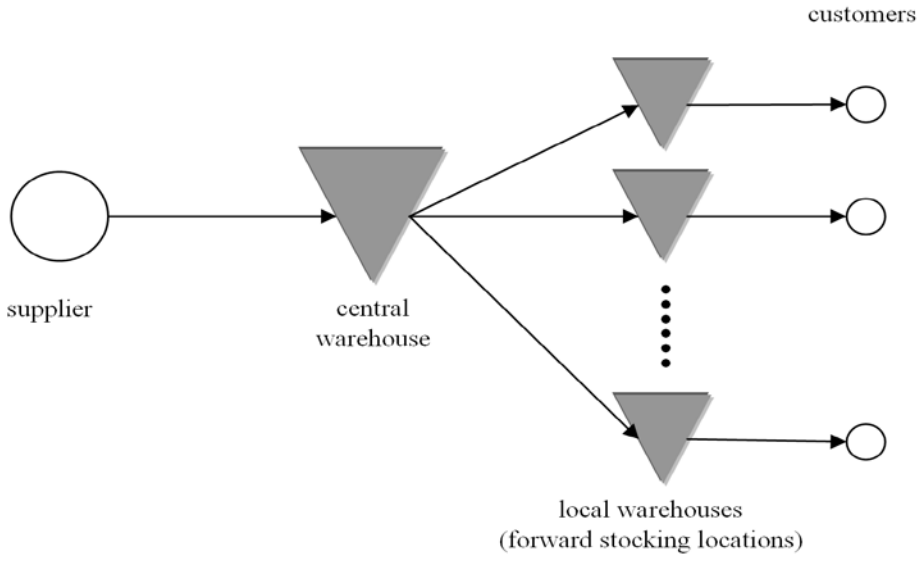

Figure 1.1 The representation of the two-echelon inventory system.

The services business is a significant portion of a technology company's portfolio and there is an understanding now that is should be targeted. In the context of telecommunications industry, not only should a service vendor be able to supply at a reduced cost and lead-time, ISPs require that failed components are replaced so the downtime to their network is kept to a minimum. The two-echelon inventory system depicted in Figure 1.1 is a typical supply chain structure that is used to supply spare parts to satisfy the requests from the ISPs. Due to stringent delivery restrictions like two-hour or four-hour delivery windows, local warehouses are utilised to supply parts to the customer equipment. The locations of the local warehouses are determined such that they are very close to the customer equipment. These local warehouses are replenished by the central warehouse. Failed parts are sent to the central warehouse by the customer, and from there they are shipped to the repair vendors. If a part requested by the customer is not delivered on time, the penalty cost is incurred per the terms of an agreement. The demand for spares tends to be low, in the order of a couple of units per year at a local warehouse; however, spares are usually expensive. Having expensive slow moving items in the system and penalty clauses make the optimisation of inventories (where to stock and how much to stock) an important task. The Penalty Cost Model is used to determine the amount of inventory required to be placed at each of the FSLs and the central warehouse to meet customer obligations. This model is discussed in more detail in chapter two.

1.2 Differential Evolution

DE is a reliable, easy to use optimisation tool. One of the advantages of DE is that it has a small number of control parameters for adjustment which adds to its simplicity. Only three parameters need to be adjusted when optimising a function, the step size otherwise known as the scale factor F, the cross-over probability and the population size. A trial and error method of selecting the best combination of the three parameters is recommended for a particular objective function and a study of the control parameters is discussed later in the thesis. DE optimisation occurs through a creation of a search space of multiple candidate solutions or vectors. Each vector undergoes the following steps in order to be considered for inclusion in the next generation of vectors.

- *Mutation:* A key characteristic of DE is how new population vectors are created for each generation. The new vectors are points creating by perturbations of the existing points. This is done by using the scale factor F to calculate a weighted difference vector between two randomly chosen vectors which is added to a third randomly chosen vector to generate a trial vector
- *Recombination:* The creation of the trial vector is through a method of recombination between the original parent or target vector and the generated mutant vector in the previous step through the process of crossover. Crossover determines how many parameters from the target and mutant vectors are used in the creation of the trial vector. Crossover ensures that the trial vector will inherit at least one parameter from the mutant vector. This maintains diversity in the newly emerging population.

- *Selection:* Selection follows the normal process inherent to most evolutionary algorithms. Each new trial vector is compared to its parent and will replace the parent in the new generation if it produces a better objective function value then its parent.

There is no bias in the way DE selects population members for consideration as this can create populations that have lost diversity. It uses a one-to-one tournament selection to determine vectors that move into the next generation. A trial vector competes against its parent and will be eliminated if it does not better the objective function value of the parent. This is the case even if the trial vector has a better objective function value then other vectors in the population. In the context of the Penalty Cost Model, our goal is to minimise the total investment over the spare parts inventory system such that we know the amount of inventory required to be placed at each of the FSLs and the central warehouse. The DE population of vectors are combinations of stocking levels for each location in the inventory system. DE iterates through the population space searching for the optimum selection of stocking levels such that the total cost is minimised. It achieves this by perturbing existing vectors of stock levels to create new vectors whose objective function values are considered. Termination of the DE optimisation occurs after a set number of iterations or convergence on the optimum solution of stocking levels.

1.3 Motivation of the Research

The main motivation behind this research is to implement DE in the context of spare part inventory optimisation in a two-echelon supply chain setting. Hence, we aim to determine the accuracy of DE in generating the optimum solutions for the Penalty Cost Model, which is a mathematical optimisation model for the problem under consideration. As the complexity of the model increases, we would like to see the effectiveness of DE and identify the scenarios in which DE is better suited for optimising the problem as apposed to other algorithms more commonly used in the field of Operations Research (OR). Finally, we would like to research how the DE algorithm scales as the Penalty Cost Model's complexity increases to determine specific DE control parameters best suited to this type of objective function. The literature referring to supply chain optimisation using DE is sparse, thus; an exploration of the benefits of applying DE to such problems is appealing.

1.4 Problem Statement and Research Questions

The research presented in this thesis studies a Differential Evolution implementation of the Penalty Cost Model to determine its comparability to the best-in-class algorithm commonly applied in the OR research domain. In order to achieve this, a number of questions need to be answered.

1. Is DE an effective optimisation tool for solving the Penalty Cost Model?
2. How does it compare with the best-in-class analytic algorithm?
3. Is the DE implementation effective at achieving the optimum solutions for a given test-bed?
4. Under what conditions does DE fail to achieve the optimum solution?

1.5 Thesis Outline

The thesis is organised as follows. Chapter 2 explores the Penalty Cost Model in more detail discussing the main components of the model and introduces Metric Approximation as a technique to determine the expected backlogs for slow moving service part items.

Chapter 3 is devoted to Differential Evolution (Price & Storn, 1995) from its inception to how it mutates existing points to generate new points as it converges towards the optimum. It outlines the advantages of DE over other evolutionary algorithms, and also explores the discrete nature of the specific implementation of DE as applied to the penalty cost model. DE control parameters and different DE algorithm variants are discussed extensively.

Chapter 4 is dedicated to the specific tasks of implementing the Penalty Cost Model using DE. A test-bed of problem setting scenarios are discussed with justification based on various strategies designed to test the ability of DE to locate the optimum solution. A review of the modifications to the Matlab package is highlighted focusing on effective boundary constraint generation, discrete mapping of continuous variables and initial population seeding.

Chapter 5 is the numerical results chapter focusing on the comparison mechanisms used to determine DE's viability as an effective optimisation algorithm for the Penalty Cost Model. An exploration of the test-bed problem settings is outlined along with the experimental setup. A review on DE's control parameters is presented characterising the key combinations of parameters that best suit the Penalty Cost Model. A comparison of DE variants is discussed. Finally, a scalability analysis of each DE variant used in this study is demonstrated with reasons for adopting particular DE variants championed under certain conditions.

In conclusion, Chapter 6 encapsulates the main points from each chapter and offer conclusions and directions for further research.

Chapter 2

2 Penalty Cost Model

2.1 Introduction

Before we describe the theory behind this services supply chain model, we provide an example that is a similar to an optimised supply chain system you will find in practice today. What differentiates the following example from a real world system is the level of complexity of the example supply chain under study. In the literature, the supply chain model we present is referred to as one warehouse, multi-retailer system.

2.2 Services Supply Chain

Consider an original equipment manufacturer (OEM), which provides telecommunications equipment to an internet service provider (ISP). OEM has contracted the repairs of fault parts to an outside repair vendor. OEM has won numerous new equipment contracts from the ISP over the last decade. However the problem of failed units in the field has always been an issue for the ISP. To mitigate the potential downtime of their network, the ISP has over-purchased units in the past to provide a buffer against failures. All failed units are returned to the OEM for repair. As a result of this, the ISP's balance sheet is showing an increase in inventory holding costs which is impacting profits. The ISP is going through the same cost cutting drive as the OEM. The OEM offers a number of spare part support services available to the ISP. The idea is to help the ISPs reduce their spare parts inventory needs by leveraging the economies of scale and economies of statistics. Note that the OEM sells equipment to multiple ISPs, so it can reduce the investment needed in spares through the reduction in uncertainty by pooling different customer (ISPs) demands, which is referred to as economies of statistics. In addition, OEM faces a greater volume of demand due to pooling multiple streams of individual ISP demand, it can benefit from lower repair and/or transportation costs, warehouse handling costs, information system costs, etc., which is referred to as economies of scale. The OEM's top service is a five-hour replacement service. Upon receiving the failure notice from the ISP, a replacement the unit will be delivered within five hours. Another service provided by the OEM is next business day delivery. Depending on the units in question for the ISP and how critical they are for the performance of their network, they choose a portfolio of services from the OEM and agree metrics and penalties to determine how well the OEM maintains the service. The penalty cost (p) is a function of the importance the part plays in the customer's network and how critical it is to the performance. In some cases, a contractually agreed price will be paid to the customer for every instance that the customer is without a replacement unit.

The ISP provides the number of units that will be supported by this new service agreement (installed base) and where they are located. The OEM already has a main central warehouse in Paris to serve European demand and uses a third party logistics (3PL) partner's (e.g., DHL) local warehouses and delivery network. The central warehouse acts as a supplier in effect to the FSLs. The FSLs act as a supplier to the customer locations where the network units are operating. The ISP has paid for the five-hour service for a number of critical items in their network. The OEM knows the number of parts in question, the location of the parts and how long it will take them to deliver the replacement from the closest local warehouse (FSL).

There are two locations inside the ISP's network that are outside the range of a five-hour delivery window from the existing supply chain of the OEM. Hence, the OEM considers this and creates two new FSLs to provide service to the customer locations. The system operates as follows. A failure notice is received from the ISP to the OEM. A replacement unit is delivered from the local warehouse (FSL assigned to that specific ISP equipment) and the failed unit is collected. The local warehouse will request a replacement from the central warehouse. All failures return to the central warehouse and from there to the repair vendor. Successfully repaired units will go back into stock at the central warehouse. This creates a closed loop of repaired parts. Parts that are eventually deemed un-repairable are scrapped, their components re-cycled, and a new part is injected to the closed system by the OEM.

2.3 Model

We present a two-echelon system; a central warehouse (CW) and multiple local warehouses (FSLs). FSLs are responsible for replenishing the customer locations. The CW has responsibility for replenishing the FSLs. We denote CW as location 0 and FSLs as locations 1,2,…..,J. Only FSLs are facing the stochastic demand of the customers, which is assumed to be Poisson distributed in this model. The inventory system provides a service for a single item. Each location follows an installation base-stock policy under continuous-review i.e., one-for-one replenishment. In other words, each customer request triggers replenishment from the CW.

We use the following notation:

Notation:

Z_0^+	=	set of nonnegative integers, $Z_0^+ = \{0,1,2,\ldots\}$;
$\boldsymbol{i}$	=	index for locations;
c	=	unit purchasing cost;
p	=	unit penalty cost associated with not satisfying demand;
L_i	=	replenishment lead time at location i, $L_i \geq 0$;
λ_i	=	arrival rate at location i, $\lambda_i \geq 0$;
S_i	=	base-stock level at location i $S_i \in Z_0^+$;

I_i = net inventory level at location i, $I_i \in \Re$;

W_0 = Expected waiting time at the CW.

At each location, we assume that the stock is allocated according to a first-come first-served (FCFS) basis.

2.4 Analysis

The optimisation problem is introduced in 2.4.1 outlining the dynamics of the system.

2.4.1 Dynamics of the Penalty Cost Model

The objective is to minimise the expected cost of the system;

$$\min_{S_0,\ldots,S_J} \quad \Pi(S_0,\ldots,S_J) = c(\sum_{i=0}^{J} S_i) + p(\sum_{i=1}^{J} BL_i(S_o, S_i)), \qquad \textbf{(2.1)}$$

where $\Pi(S_0,\ldots,S_J)$ is the cost function and $BL_i(S_o, S_i)$ is the expected number of backlogs at the FSL i.

FSLs face the stochastic demand from the customers. The failed unit is immediately replaced by a serviced part from stock if it's available. If it is not available, the demand for a serviced part is added to the backlog. The expected number of backlogs is a function of both the base stock level at the CW and that particular FSL, and can be calculated as follows:

$$BL_i(S_0, S_i) = \lambda_i(1 - \Pr\{(D_i(L_i + W_0) < S_i\}), \qquad \textbf{(2.2)}$$

where $D_i(L_i + W_0)$ is the random variable denoting demand at location $\boldsymbol{i}$ over time duration $L_i + W_0$ with W_0 being a random variable. The probability term in (2.2) is the probability that arriving demand at location $\boldsymbol{i}$ is fulfilled immediately from on hand stock. Hence, one minus this probability corresponds to the probability of not satisfying the arriving demand.

The waiting time at the CW (W_0) is calculated using an approximation technique referred to as METRIC approximation in the literature [2]. See Sherbooke *et al* (1968). Metric approximation replaces the random variable W_0 with the expected waiting time at the CW. As a result of applying metric approximation to the model, solving (2.1) provides an approximation of the stock levels required instead of the exact levels.

To calculate the expected backlogs across all FSLs, Let λ_0 denote the arrival rate at the CW

$$\lambda_0 = \sum_{i=1}^{J} \lambda_i, \qquad \textbf{(2.3)}$$

where λ_0 is the sum of all arrival rates over the FSLs.

METRIC approximation uses the following relationship for the calculation of W_0:

$$W_0 = \frac{BL_0(S_0)}{\lambda_0} \quad \textbf{(2.4)}$$

where $BL_0(S_o)$ is the expected backlog level at the CW. Note that the net inventory level at the central warehouse is

$$I_0 = S_0 - D_0(L_0),$$

where $D_0(L_0)$ is the demand faced by the CW over its replenishment lead time.

Let $a^+ = \max\{a,0\}$ for any $a \in \Re$. For I_0, we can write the following equality:

$$I_0 = (I_0)^+ - (I_0)^-.$$

Thus, $(I_0)^- = (I_0)^+ - I_0$, and

$$\begin{aligned} E[(I_0)^-] &= E[(I_0)^+] - E[I_0] \\ &= \sum_{u=0}^{S_0} (S_0 - u)\Pr\{D_0(L_0) = u\} - (S_0 - L_0\lambda_0) \end{aligned}$$

Note that $BL_0(S_o) \equiv (I_0)^-$. Using this identity and substituting the expression above into (2.4) yields

$$W_0 = \frac{\sum_{u=0}^{S_0} (S_0 - u)\Pr\{D_0(L_0) = u\} - (S_0 - L_0\lambda_0)}{\lambda_0}. \quad \textbf{(2.5)}$$

The system minimises the expression (2.1). It provides the overall total investment cost for a set of input parameters. The decision variables for this function are the base stock levels at the CW and FSLs, the arrival rates and lead-time at the FSLs and the CW lead-time. The penalty cost for the expected number of backlogs at the CW is given by (2.6).

$$p\sum_{i=1}^{J} BL_i(S_o, S_i) \quad \textbf{(2.6)}$$

Chapter 3

3 Differential Evolution

3.1 Introduction

Differential Evolution (DE) is a powerful, easy to use, fast, reliable global optimisation algorithm that is a tool for tackling difficult optimisation problems. Since it's inception in 1995, DE's reputation as an effective optimisation algorithm has grown. It has been applied to various different optimisation problems and different variants of DE have been proposed [3]. Its origins are in the Genetic Annealing algorithm (Kenneth Price, 1994 Dr. Dobb's Journal DDJ). Genetic annealing is a population based, combinatorial optimisation algorithm that is used to find a solution via thresholds in a space deemed too large for ordinary search methods. After publication in DDJ, Dr. Rainer Storn investigated the possibility of solving the Chebyshev polynomial fitting problem. The initial solution was found but convergence was slow. Modification of the Genetic Annealing algorithm using floating point encoding and arithmetic operations as well as discovering the differential mutation operator gave rise to the first iteration of DE. Storn suggested creating separate parent and child populations to accommodate parallel processing. DE was presented by Storn and Price in 1995 [4] followed by [5,6,7] and was well received at the IEEE International conference on Evolutionary Computation. Another article published for DDJ in April 1997 introduced DE to a large international audience. After another publication with extensive empirical evidence of DE's robustness [8], many optimisation researchers became aware of DE's potential. More light was shed upon the inner workings of DE in 2002 [9] with an article on the analysis of the critical values for control parameters of DE. The paper deals with one problem of premature convergence which is common in evolution strategies and the choice of control parameters that prevent premature convergence or stagnation

3.2 Motivation for using DE

DE is a population-based stochastic method for global minimisation of objective functions. Objective functions are a way of quantifying everyday real world problems that describe properties that need to be minimised to obtain some particular outcome. Often objective functions can have many parameters that will influence the property that is being optimised. Objective functions will have a number of characteristics that help determine the how well it will be optimised. Are input parameters continuous, discrete or both? What is the dimensionality of the objective function? , is the function multi-modal in nature, are there constraints placed on the objective function? DE is used for optimising multi-objective functions that often have objectives that are in conflict with each other. One of its main

attractions is its ability to find optimal solutions without the need to compute derivatives. It is, in summary, multi-point derivative-free optimiser.

3.2.1 Derivative-free techniques

Derivative-free techniques are known as direct search algorithms and rely on heuristics and conditional branches to find the optimum. Using direct search, there is a selection phase during which a proposed move is either accepted or rejected. One such direct search method is the brute force method or Enumeration. However, this technique is rarely used to optimise functions with a significant number of continuous parameters. All methods need to consider a suitable step size when searching through some grid space. If the step size is too large, the optimum value could be missed; if the step size is too small, it will be computationally too expensive to proceed. The dimensionality of the problem is why brute force methods are not considered. One such method that over comes dimensionality is the random walk method (Gross & Harris, 1985) which samples the objective function at randomly generated points and has a greedy selection criterion. Nevertheless, choosing a good step size figure is still an issue for the Random Walk method. The Hooke and Jeeves method [10] is a one-point direct search that tackles the step-wise problem. It uses its own step size to search for trial points along each co-ordinate axis. If the step size is deemed too large, a smaller step size is used and the procedure repeats. While the Hooke and Jeeves method is more effective at searching a population then Brute Force or Random walk, the step size is always reduced in the search for new trial points. An algorithm with no possibility of increasing the step-size can get trapped in a local minimum. For unimodal objective functions, this is okay but as most real world problems are multi-modal in nature, global optimisation of a function is required.

3.2.2 Starting point problem

As multi-modal objective functions have multiple minima, they pose a starting point problem. To find the global optimum of an objective function with multiple minima, a good starting point in the vicinity of the optimum is essential. A method is required that increases the chance of sample points moving to another local minima of attraction. It is possible using the Random Walk method in conjunction with simulated annealing (SA) [11]. SA looks for points on the objective function's surface by modifying the greedy selection criterion to accept some uphill moves while also continuing to accept all downhill moves.

While there is still an issue with step size for SA as a search criterion, it has been used as a substitute for greedy criterion in direct search methods like [12]. However, SA is not particular adept at continuous function optimisation due to step size. Multi-start techniques are another way to sample an objective function space. Each sample point is an initial point for a greedy search of the local landscape. There are drawbacks to this method attempting to calculate how many starting points are

required. Also, if a lot of starting points are close together, they may all lead through a greedy search to the same local minimum. Various clustering algorithms (Torn & Zelinkas 1989; Janka 1999) have been applied in tandem with mutli-search techniques to identify clusters of points that belong to the same local minimum. Base points will be supplied for each cluster which will serve as a base point for local optimisation. Multi start techniques are multi-point, derivative-based methods. Differential Evolution belongs to a group of methods that are multi-point, derivative-free. Along with Evolution Strategies (ES) and Genetic Algorithms (GA), they all attempt to find better solutions through a process of recombination, mutation and selection of the best candidates (Survival of the fittest). ES, GA, DE are often referred to as evolutionary algorithms (EA) but there are differences.

ES methods like Nelder and Mead polyhedron search 1965 and Controlled Random Search Price 1978 are good at optimising continuous objective functions using floating point encoding of parameters while GA's are often better at optimising combinatorial problems using bit string encoding. Modifying GA's to use floating-point encoding for continuous parameter optimisation turns the algorithm into an ES. Like the multi-start algorithm, ES samples the objective function space with many different points. Unlike the multi-start algorithm whose points evolve in isolation, ES points influence each other through the means of re-combination. ES algorithms are some of the best at global optimisation of a search space. Like nearly all EAs, DE solves the starting point problem by randomly sampling the objective function at multiple initial starting points. DE generates new points for consideration that are perturbations of existing points. How DE differs from other ES methods is the newly generated points are not samples from a predefined probability density function or reflections of existing points (Nelder and Mead polyhedron search 1965, Controlled Random Search – Price 1978).

3.3 Differential Evolution Algorithm

DE is more likely to be affected by premature convergence then other evolution strategies as it does not employ outside information when optimising. In 2005, the term Contour matching was coined [1]. This describes the self steering properties of DE from initialisation with competing minima and multi-modal search towards the optimal minimum with steps sizes appropriate for a local search.

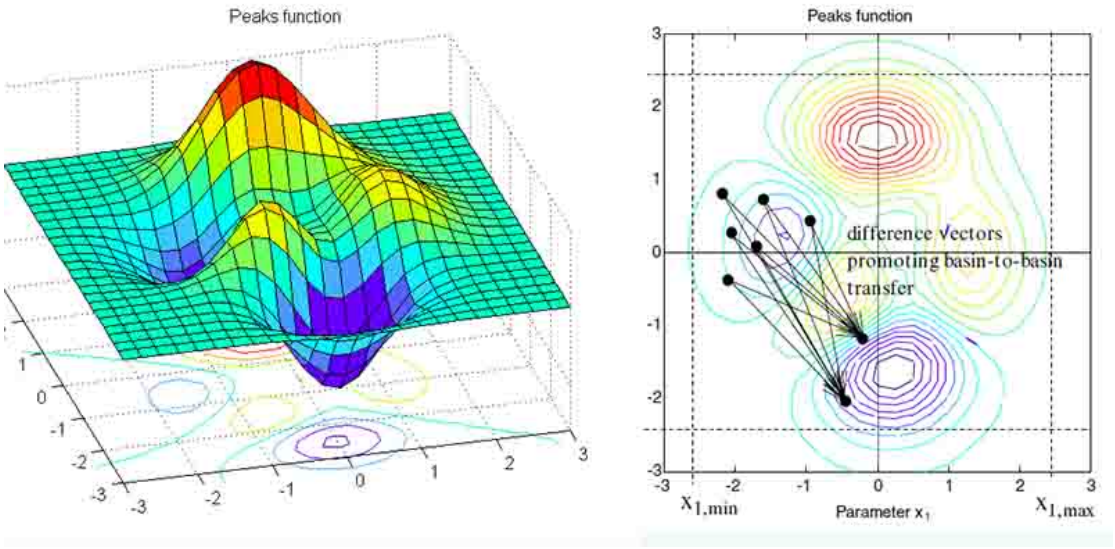

Figure 3.1 Peaks function and illustration of difference vectors that promote transfer of points between local minima [1]

Contour matching means that the vector population adapts such that promising regions of the objective function surface are investigated automatically once they are detected. Part of the investigation is the promotion of basin-to-basin transfer [13]. Figure 3.1 illustrates that DE yields a certain amount of difference vectors that can generate new trial points in the lower basin of attraction when the base points originated from another basin of attraction [1]. Diversity of the population is critical for basin to basin transfer. The number of population members (NP) is a control parameter that is set for the duration of algorithm. One of the main aspects of DE is the generation of new vectors which are perturbations of existing vectors. The perturbed vectors are generated from an NP*(NP-1) nonzero difference vectors of the population instead of using a predetermined probably density function. The new points are reflections of existing points which are binomially distributed. This fosters a wide search for the global minimum. As the difference vector distribution decreases through each generation, we see DE self adaptively explore the basin that is covered by the decreasing vector distribution.

After initialisation of the population, DE applies mutation to the population of vectors to create a mutated population of vectors. This mutated population is combined with the initial target population to create the trial population of vectors. As part of the generation of trial vectors, DE employs a technique known as Crossover (CR). CR is a user-defined probability that controls the rate of parameter values that are copied from the mutant vector. To determine which vector will provide the parameter for the trail vector, CR is compared to a random number generator between [0, 1], if the random number is less then or equal to CR, the mutant vector supplies the parameter for the trial vector; otherwise the target vector supplies the parameter. After the trial population of vectors is generated, DE applies selection to both the trial and target populations. The vector with the lower objective function cost is marked for the next generation of population vectors. By comparing each trial vector with its corresponding target vector of the same index, DE more tightly integrates the recombination and selection process. It also ensures that each target vector is compared only once per generation. The vectors that are marked by the pair-wise comparison become parents of the next

generation. All target vectors will compete against randomly generated trial vectors until the new population for the next generation is created. This process will continue until the optimum is found or a termination criterion is reached.

3.3.1 Initialisation

One requirement for DE to optimise an objective function is that the function's landscape needs to have a population that is distributed through out the problem space.

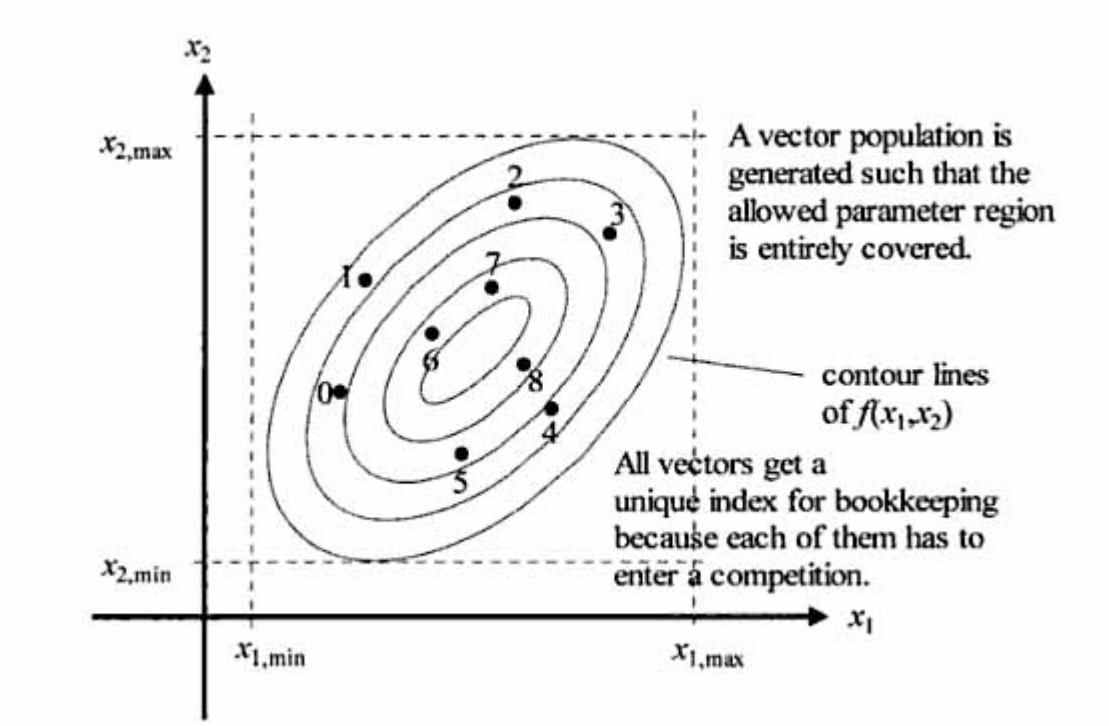

Figure 3.2 Initialisation of DE Population.

Initial bounds are provided as user input to cover an area that will contain the global optimum. Most real-world problems will have parameters whose bounds are seldom equal. Where the difficulty arises is if the bounds for a specific parameter are not known. We enter a situation whereby we must use far initialisation to tackle the lack of bounds. With far initialisation, there is the possibility that the optimum lies outside the initial bounds. It is critical that bounds are set sufficiently enough to cover the problem space. The initial spread of points can converge on a local minimum that is not the global minimum if F is too small. If competing minima are too far apart, DE cannot use difference vectors to jump from one minimum to the next. Most real world objective functions will have parameters that correspond to some physical or logical constraints that will suggest bounds that are acceptable to use. Uniform random distributions are effective distributions to use for initialisation especially when the optimum location is unknown. In general, any distribution that contains some random distribution of points across the problem space is good to use for initialisation. If some information about the optimum is known, a Gaussian distribution can be applied to distribute the points in a multi-normal

fashion. If the objective function is multi-modal in nature, uniform random distributions should be used to distribute the initial points widely enough to encompass the optimum.

3.3.2 Mutation

A mechanism for evolving the population of vectors is essential. There is the possibility that re-selection of vectors already chosen can occur along with other vectors being omitted from the search. Vectors that are not chosen are deprived of passing on potential diversity to the next generation. Re-selection of vectors causes the potential to loose diversity in the next generation due to over sampling of the same vector. DE ensures that this does not happen by comparing vectors from competing populations by their index. The target vector, *i*, specifies the vector that the mutant vector will be recombined with to form the trial vector. Indices r0, r1 & r2 are used to determine which vector parameters create the mutant.

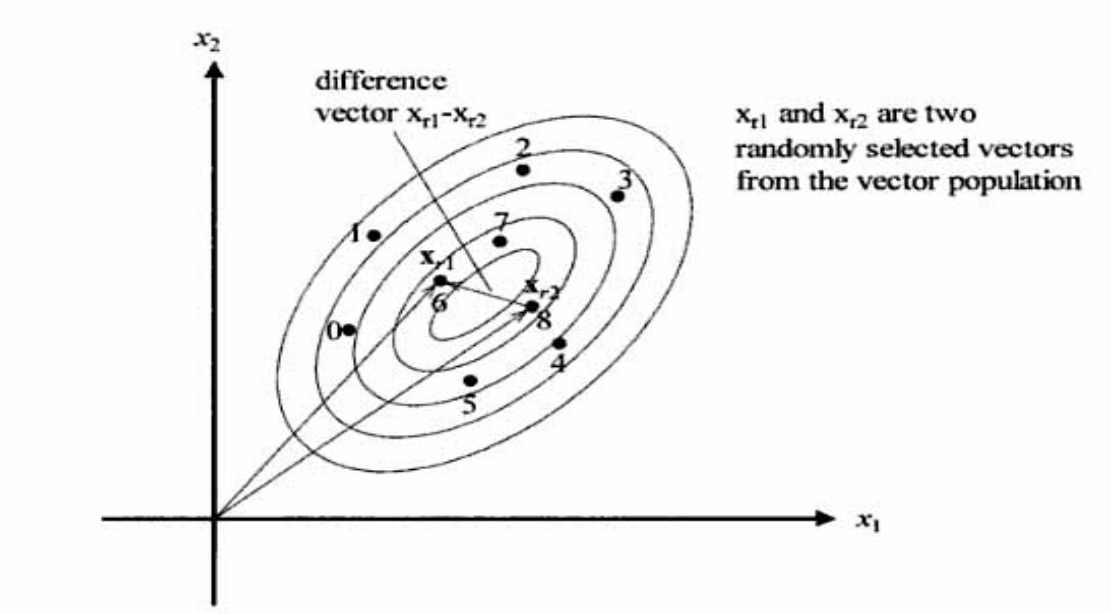

Figure 3.3 Perturbation of two vectors.

The base vector index refers to r0. The difference vector indices refer to r1 & r2. The base vector index represents the vector that will have the scaled difference ($F \in [0,2]$) vector added to it along with a third randomly chosen vector to form the mutated vector, u. (Fig. 3.5)

$$u_0 = x_{r0} + F \bullet (x_{r1} - x_{r2}) \quad \textbf{(3.1)}$$

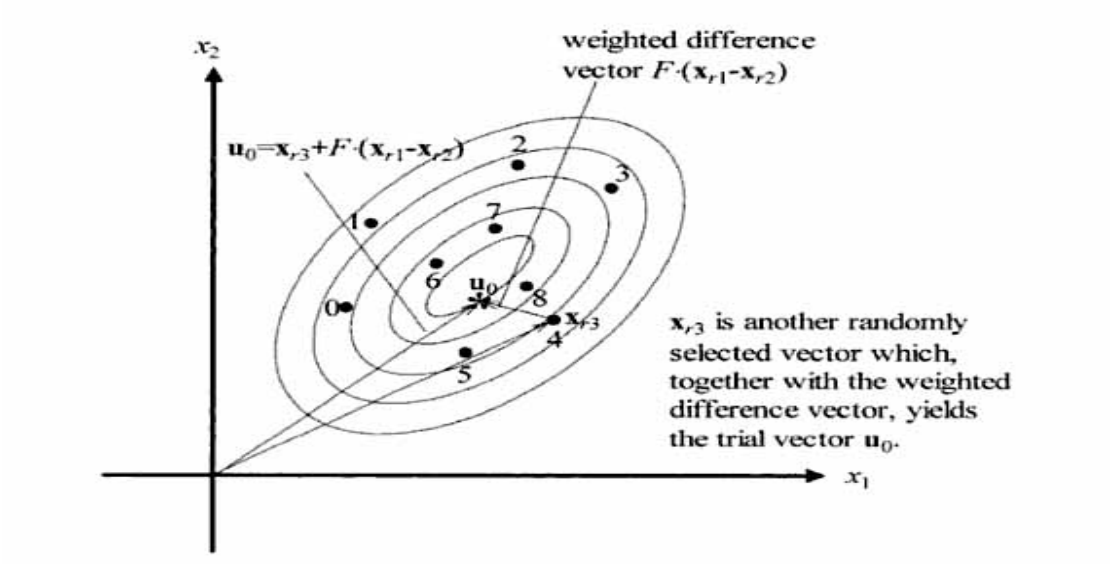

Figure 3.4 Mutation.

The mutation scale factor F has a range of [0, 2] although the results generated from this research do not use an F value > 1.1 as a scale factor. This is discussed in chapter 5, review of DE control parameters. In general, solutions with F > 1 are more time consuming and less reliable then F < 1.

3.3.3 Recombination

Recombination of vector parameters creates one or more trial vectors for pair-wise comparison with the target vector. The target vector is already known from the current population. Discrete recombination is known as crossover (CR) and is an operation in which trial vector parameters are copied from randomly selected vectors. Discrete recombination is used in classic DE where at least two populations of vectors are maintained, the current population being replaced by the newly formed next generation of vectors. This is a family survival selection criterion. In contrast to this, continuous recombination is another method where only one population is maintained and newly formed vectors replaced randomly chosen vectors in the current population and are immediately available to be used to generate new trial vectors for comparison. Nonetheless, continuous recombination is not best suited for data that is symbolic or binary in nature.

In order to generate a new generation of vectors, the trial vector is created using CR on the target and mutant vectors respectively. The most popular method of CR is binomial CR described as follows:

$$\mathbf{u}_{i,g} = u_{j,i,g} = \begin{cases} v_{j,i,g} & \text{if } \left(\text{rand}_j(0,1) \le Cr \text{ or } j = j_{\text{rand}}\right) \\ x_{j,i,g} & \text{otherwise.} \end{cases} \quad \textbf{(3.2)}$$

In binominal CR, $rand_j[0,1] \in [0,1]$ denotes the j-th evaluation of a uniform random number generator. $C_R \in [0,1]$ is the user defined input CR probability. CR ensures that the trial vector will inherit at least one variable from the mutant vector. There are a number of different CR techniques which can be applied along with binominal CR, one-point CR, n-point CR and exponential CR. In this case, binominal CR is preferred as is does not exhibit a representational bias when selecting parameters for inclusion in the trial vector. Representational bias is inherent in n-point CR since the way parameters are ordered in vectors affects the algorithms performance.

Comparing n-point CR to binominal CR, each parameter regardless of position in the trial vector has the same probability of inheriting its value from a given vector. For example, CR = 0.3 and CR = 0.7 produce a vector that on average inherits 30% from one vector and 70% from another. In binominal CR, comparing CR to $rand_j[0,1]$ determines the source of trial parameters. To begin, DE will select a parameter at random from the mutant to ensure that the trial vector will not be an exact copy of the target vector. Based on a uniform random number generation value $rand_j[0,1] \leq Cr$, the parameter will be provided by the mutant vector; otherwise, the target vector is the source. When CR is close to 0, DE minimises disruption by changing a few parameters of a vector at a time, while when CR is close to 1, this favours exploration as most parameters will be taken from the mutant vector.
In the study of the control parameters [7] for DE, it found that a low CR is the most effective for optimising decomposable functions where as CR ~ 1 are effective for functions that are not decomposable.

3.3.4 Selection

There are two stages in the evolutionary process where selection can be applied to a population. Certain GAs [14] employs a selection criterion that is biased towards the parent. Typically, vectors that have the best objective function values are assigned the highest selection probability making them more probable for mating in the next generation. This is similar to breeding of animals by improving certain characteristics of the animal's offspring i.e. racehorses. Instead of selecting potential mates based on objective function values, evolution strategies (ES) and DE select vectors with equal probability. DE randomly selects base vectors for recombination with no reference to their objective function values. There is a difference between this method and GAs. Most EAs will employ some selection pressure on potential mates for recombination or survival. GAs will typically bias selection in favour of better vectors for recombination and survival whereas DE only applies pressure when picking survivors for the next generation i.e. the objective function value pair-wise comparison. For selection in the next generation, there are different survival criteria that can be explored; age only, objective function value, age and objective function value. Using objective function value only, the algorithm retains the best-so-far solution which is known as elitism [15]. DE has a one-to-one selection policy and holds NP knock out competitions to determine the next generation. NP refers to the number of population members.

DE shares similarities with Particle Swarm Optimisation (PSO) in this regard adopting a one-to-one tournament selection. In the PSO algorithm, it has a velocity update formula that is similar to DE variant DE/target-to-best/1. It is a form of binary, deterministic one-to-one selection. Any vector that loses the single competition is eliminated and any vectors that win move forward into the next generation. In DE, the best performing vector at the i-th position is simply the i-th vector in the current population. The trial vector replaces the best-so-far vector with the same index only if it has an equal or lower objective function value. Comparing each trial vector to the best performing vector at the same index ensures that the best vector is retained at each index but also the best-so-far vector at any index. However, with one-to-one selection, a trial vector will be rejected even if its cost function value is better then most of the current target population but not the target vector it is competing with.

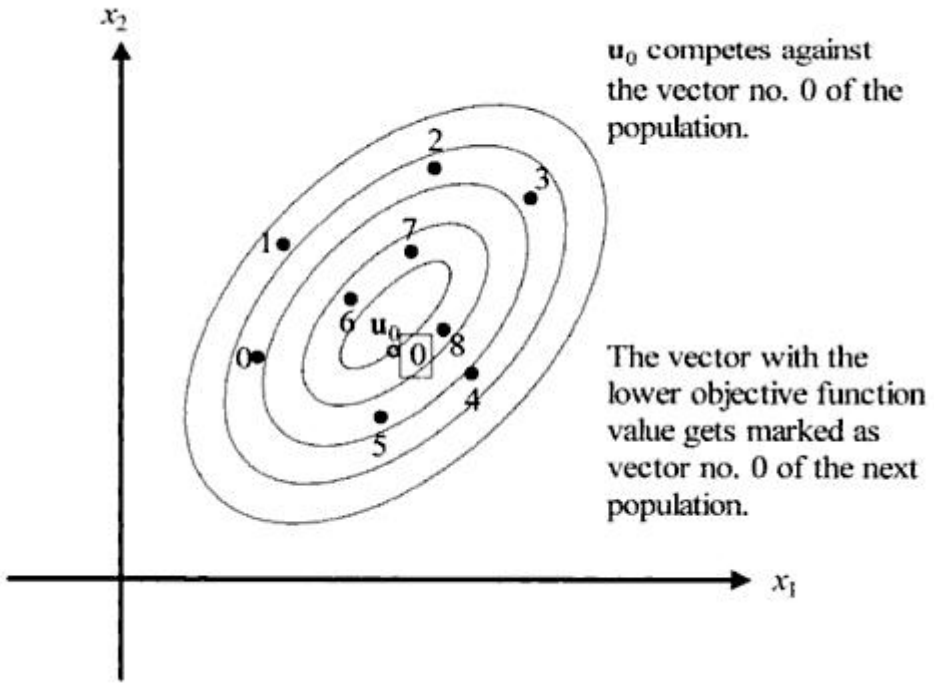

Figure 3.5 Selection. Vector u has a lower objective function value then the target vector number 0 so it moves forward to the next generation.

When the objective function is known to be multi-modal in nature, global selection is taken to ensure all members of the population receive information about the best solutions. Comparing global selection to local selection limits the potential to get trapped in a local minimum. With local selection, the population is divided into clusters or sub-populations which evolve in isolation. Information between clusters does not pass between members and as such, there is a risk of stagnation.

3.3.5 Termination

When is the global optimum achieved? It depends on the objective function. If the function is a multi-objective function, objectives can conflict with each other. Satisfying one objective often leaves another unsatisfied. Pareto improvement is required whereby changing one objective does not make

another worse off. Often as a user defined input, the user can limit the number of iterations of the algorithm. This is a trial and error approach in that a sufficient number of iterations are required to ensure the best known results are returned. Another method for termination is when the objective has been met. In some objective functions, the minimum value can already be known. For example, some minimum value may already be known about a function meaning any subsequent value found under this value is not considered. This is also a method used when working with functions where the minimum is known. If the best-so-far vector's objective function value is known to fall within a certain tolerance of the global minimum, termination is met. Also, human monitoring can determine when optimisation is over. Feedback provided by the objective function can determine that no further optimisation is possible.

3.4 DE Algorithm

The code used in this thesis is the DeMat Matlab package [1]. A more detailed description of the main functions and files is given in section 4.3. Below is pseudo code of the classic implementation of Differential Evolution. As described later in the thesis, there are a number of different variants of the algorithm that adjust how new members are selected for inclusion in the next generation.

// Inputs $Population_{Size}(NP)$, $problem_{Size}(D)$, $ScaleFactor(F)$, $Crossover(CR)$
// Output: S_{best}
Population ← InitialisePopulation(NP, D)

EvaluatePopulation(Population)
S_{best} ← GetBestSoFar(Population)

While(TerminationNotMet())
    NewPopulation = ""
    For ($P_i \in$ Population)
        For(j=0; j< D ; j++) // generate a trial vector
            If(rand(0,1)<= CR or j==RandomPosition(NP))
                $S_{j,i} = P_{j,r0} + F * (P_{j,r1} - P_{j,r2})$
            Else
                $S_{j,i} = P_{j,i}$
            End
        End
        If (Cost (S_i) <= Cost(P_i))
            NextGeneration ← S_i
        Else
            NextGeneration ← P_i
    End
    Population ← NextGeneration
    EvaluatePopulation(Population)
    S_{best} ← GetBestSoFar(Population)
End
Return(S_{best})

3.5 DE Control parameters

In DE, diversity of the population is critical to avoid premature convergence. In general, selection tends to reduce the diversity of the population whereas mutation increases it. DE has three control parameters that maintain this diversity, the mutation scale factor of the difference vector (F), the crossover control parameter (CR) and the population size (Np). These are fixed values for the duration of the algorithm. There is research into the self adaptive control parameters [16] that monitor and adjust the parameters on the run as the population evolves but for the purpose of this review, we do not apply adaptive control parameter setting. There are two methods for parameter setting, parameter tuning and parameter control. Self adaptive control parameters fall into the parameter control category. In the experimental results we highlight how parameter tuning was taken to find the optimum control parameters for the penalty cost model. Parameter tuning is used in a trial and error fashion to find good starting point control parameters that are used in the algorithm. Fine tuning the parameters based on initial results is the feedback to tune the parameters for the main run of the algorithm that best suits the test-bed.

3.5.1 Scale factor F

Maintaining a good F is crucial to counteract the selection pressure. In [9] Zaharie shows that there are upper and lower limits of F and it is not recommended to use an F outside of this threshold. Zaharie's study provides an algorithm to effectively calculate the lower limit.

$$F_{crit} = \sqrt{\frac{\left(1 - \frac{P_{CR}}{2}\right)}{Np}} \qquad \textbf{(3.3)}$$

F_{crit} provides a lower limit of F in the sense that values smaller then this will induce convergence even on a level objective function surface. Objective functions rarely have surfaces that are flat. F must be greater then F_{crit} to counter the additional reduction in variance that selection delivers. Classic DE employs an F value that is constant as the algorithm iterates. There are other modifications of DE that can introduce a method of randomising the F difference vector scale factor. Keeping F constant is effective in the sense that no function that has been solved required F to be randomised.

Randomising F can prove advantageous. Creating a random variable F widens the amount of potential difference vectors that can be picked to create mutants. This is an excellent option to use for populations that are small or symmetrically distributed. DE requires a population to have enough diversity so as not to stagnate. When DE stagnates, it cannot find better solutions because no combinations of vector differences and vectors provide a better solution. Randomising the scale factor

F is a good way of increasing the pool of potential trial vectors while minimising the impact of stagnation without the need to increase the population size. In the experimental results, there are two algorithms tested that introduce jitter and dither into the generation of F as the scale factor. Jitter is used to generate a new value of F for every parameter in the vector. Alternative to this is dither in (3.4) when a new F scale factor is generated for each vector.

$$F_{dither} = F + rand_i(0,1) \bullet (1 - F) \quad \textbf{(3.4)}$$

for each difference vector, *i.* Dithering DE does not significantly depart from classic DE in that the F scale factor generated is applied to all components of the difference vector. Jittering the scale factor F is a significant change to classic DE in that all components of the difference vector are multiplied by a different F scale factor and this changes the scale of the differential but also its orientation.

$$F_{jitter,i} = F + rand_j(0,1) - 0.5)) \quad \textbf{(3.5)}$$

This makes DE Jitter different to classic DE with F as a constant value. Part of Zaharie's study [9] included various tests done comparing classic with jitter and dither versions of DE. It was estimated that jitter DE was the fastest technique over the full range of CR as a random real number in the range [0,1] . In terms of the number of function evaluations, all three methods of DE require the same amount of evaluations. Classic DE and dither perform similarly but dither requires a larger population size.

3.5.2 Population members NP

DE requires preset parameter bounds to generate an initial population that is uniformly distributed throughout the problem space. This population is a D-dimensional matrix of vectors with NP population members. The general acceptable rule for NP is 10*D. The number of population members NP is also not very critical. Depending on the difficulty of the problem NP can be lower than 10*D or must be higher than 10*D to achieve convergence. NP does not change as the algorithm iterates towards the optimum.

3.5.3 Crossover probability CR

The cross over probability (CR) is a real numbered value [0,1] and manages the recombination of parameters to create the trial vector. Higher values of CR mean the parent target vector has less influence in passing parameters to create the trial vector. [1] Another important characteristic of CR is that only high values of CR guarantee the contour matching properties of DE are applied. However, [1] also shows that contour matching is lost when CR = 1 i.e. strong crossover and that DE has a tendency to search along the main parameter aces which is beneficial to separable objective functions.

3.6 DE notation

Different variants of DE have been created as new optimisation problems are investigated. Various objective functions have populations that are non-linear, multi-modal, small number of parameters and so on. A notation was designed to allow the user to understand what type of variant is employed as the main DE algorithm for optimisation. Classic DE is as follows:

3.6.1 DE/rand/1/bin

$$u_{j,i} = x_{j,r0} + F \bullet (x_{j,r1} - x_{j,r2}) \quad \textbf{(3.6)}$$

where $i, r_0, r_1, r_2 \in \{1,2,.....,N_p\}$; r0,r1,r2 are randomly chosen and $i \neq r_0 \neq r_1 \neq r_2; x_{j,best}$ is the best individual vector in the j-th generation. The scale factor F is a real number in the range [0,2]. For the experimental results discussed in chapter 5, the scale factor F range was [0.5,1.1]. This is the most popular variant of DE and is known as classic DE. The notation **DE/rand/1/bin** is described as follows, DE Differential Evolution, "rand" refers to vectors that are selected at random to create mutant vectors, "1" is the number of pairs of solutions chosen and "bin" refers to the recombination stage creating the trial vector using a binomial recombination technique. Another variant used in the experimental stage is

3.6.2 DE/local-to-best/1/bin

$$u_{j,i} = x_{j,i} + F \bullet (x_{j,best} - x_{j,i}) + F \bullet (x_{j,r1} - x_{j,r2}) \quad \textbf{(3.7)}$$

This DE variant computes the difference between the i-th member and the best-so-far member of the current population. This method attempts to balance robustness with fast convergence and is a popular choice in most studies of DE.

3.6.3 DE/best/1/bin with jitter

$$u_{j,i} = x_{j,best} + F_{jitter} \bullet (x_{j,r1} - x_{j,r2}) \quad \textbf{(3.8)}$$

Using equation 3.5 to add jitter for the scale value F for each parameter in the j-th vector, this algorithm always selects the best-so-far vector as its base vector in addition to 1 scaled vector difference creating the trial vector by uniformly crossing the resulting mutant with the target vector. In this algorithm, the base vector always has the lowest objective function value in the current target population and adds

increased greediness to the search process. This results usually in faster convergence, reduced odds of stagnation but a lower probability of success.

3.6.4 DE/rand/1/bin with per-vector-dither

$$u_{j,i} = x_{j,r0} + F_{dither} \bullet (x_{j,r1} - x_{j,r2}) \quad \textbf{(3.9)}$$

Using equation 3.4 to apply dither on a per vector basis, this algorithm is closely related to classic DE but with more robustness.

3.6.5 DE/rand/1/bin with per-generation-dither

Same strategy as equation 3.9 put dither is applied once per generation basis instead of each vector.

3.6.6 DE/rand/1/bin either-or

$$u_{j,i} = x_{j,r0} + 0.5 \bullet (F+1) \bullet (x_{j,r1} + x_{j,r2} - 2 \bullet x_{j,r0}) \quad \textbf{(3.10)}$$

Expression 3.10 is used after a random number check. If rand < 0.5, classic DE in equation 3.6 is applied, else equation 3.10 is applied.

3.7 Discrete Differential Evolution (DDE)

The previous section describes different variants of the classic DE algorithm that have been applied to many numerical optimisation problems. DE was developed as a non-linear continuous optimisation algorithm. The objective function under study contains discrete decision variables; thus, the application of DE to an objective function that is discrete in nature is limited. As a result, it is important to accurately convert DE continuous nature to a discrete representation in order to optimise the objective function. Due to the nature of the problem under study, using some of the readily available rounding functions to encode floating point numbers to integer values like “floor” “round” are unsuitable. The discrete values to be minimised are inventory stock levels and incorrect levels of plus or minus 1 can lead to an exponential increase in the total investment cost and penalty costs. Taking this into consideration, simply rounding to the nearest integer value is unsuitable. The process is randomised by applying a uniform random number $rand\ (0,1) \leq 0.5$ to decide the procedure of rounding floating point solutions to discrete values. Novel DDE approaches have been proposed to solve discrete optimisation problems like DE with local search based on Lin Kernighan-Heulsgaun (LKH) method [17], DE with local search based on Variable Neighbourhood Search (VNS) [18],[19] can be enhanced with methods that improve it

Chapter 4

4 Differential Evolution Penalty Cost Model

4.1 Introduction

In this chapter, we will discuss the DE implementation of the Penalty Cost Model outlining the problem settings required by the model and the necessary software enhancements to implement it using DE. We illustrate various problem settings that mirror real world supply chain systems implementing the Penalty Cost Model and what we would expect optimum solutions to look like. This chapter is presented as follows;

- Penalty Cost Model Scenarios.
- DE code.
    - Package description
    - Boundary Constraints
    - Discrete Values
    - Seeding initial population

4.2 Penalty Cost Scenarios

The objective of the optimisation problem under consideration is to minimise the sum of the investment cost and the expected penalty costs. Penalty cost is defined as the cost applied by the customer in the event that a request for a replacement unit is not fulfilled immediately. As part of the contract negotiation between the customer and the OEM, a penalty cost is factored in as an assurance by the company that they will maintain adequate stock to help maintain the customer's network. The trade-off is between the inventory investment cost and the expected penalty cost. On one hand, no investment in spares yields minimum investment cost, but the penalty costs would be maximised since all the customer demand would be backlogged. On the other hand, investing infinitely many spares would minimise the penalty costs since all customer requests would be fulfilled from on-hand stock, but the investment in spares would be infinite.

In order to test the effectiveness of DE's ability to find the optimum Penalty Cost solutions, a series of penalty cost scenarios are developed for testing purposes. The Penalty Cost Model has various problem settings that describe the overall system. There are a number of opportunities to develop testing scenarios for the model and for DE is find the optimum solutions.

The parameters of the Penalty Cost Model are J (the number of FSLs), c (unit purchasing cost), p (unit penalty cost), L_i for $i = 0,...,J$ (replenishment lead times at all stocking locations) and λ_i for $i = 1,...,J$ (arrival rates at the FSLs). We define a combination of the values of all the parameters a scenario. For example, $J = 2$, $c = 1000$, $p = 10000$, $L_0 = 20$, $L_1 = 2$, $L_2 = 1$, $\lambda_1 = 1$ and $\lambda_2 = 0.1$ describes a specific instance of the Penalty Cost Model, and be referred to as a scenario or problem instance. As part of the full factorial test bed described later in this chapter, we fix the number of retailers, the unit purchasing cost and the FSL lead times, and vary three of the variables to various combinations of low, medium or high. Taking penalty cost as the first variable, our knowledge of the system implies that if the penalty cost is high, it is imperative that enough stock is available to meet any potential requests for replacements. Taking that into consideration and looking at the CW lead-time as another tuneable variable, a high CW lead-time means a longer repair cycle. The third variable which can influence the model is the vector of arrival rates at the FSLs. This is the rate the each FSL receives requests for replacement parts per calendar day. An arrival rate of one unit per day is considered extremely high in terms of service supply parts as this equates to 30 units per month.

Applying different combinations of low, medium, high to the model parameters is the basis of the testing scenario. A particular scenario may include a high penalty cost coupled with a high CW lead-time and a high arrival rates at the FSLs. When the optimum solution for this scenario is reviewed, it provides interesting information about the performance of the model. The solution will point to high levels of stock at the FSL to mitigate the high arrival of requests for replacement units. Due to the extensive repair lead-time, extra buffer stock is required at the CW to replenish the FSL. As the CW lead-time is decreased, the ability to return repaired units into the system increases. Thus, the need to maintain a buffer stock at the CW is decreased. As the arrival rates drops, the FSL stock levels will follow suit. It is expected that the FSL will still maintain a higher level of stock compared to the CW due to the high penalty cost. Adjusting the penalty cost down under the same conditions will decrease the FSL stock levels as the scenario moves towards holding stock at the CW. The arrival rates at individual FSLs are also tuneable to have a mixture of low, medium, high across the FSLs. Scenarios of this type are considered the most challenging to obtain the optimum solution. Each FSL has different requirements in terms of stock levels and the CW must maintain enough buffer stock to support the FSLs.

4.3 DE Matlab Package

The code used in this thesis is the DeMat Matlab package [1]. The program of the package is depot.m. It controls DE mutation, selection, crossover, maintaining boundary constraints and the output of results. The package contains a number of files that are called by the engine. Rundeopt.m is the main management file that contains DE control variables, population sizes, number of iterations, and bound constraints. This file is enhanced to allow the delivery of different testing scenarios to the main engine. This Penalty Cost Model is implemented using Objfun.m. The file takes vectors of stock levels from the CW and FSLs as input along with the FSL arrival rates and lead-times to compute the total cost. Another critical file is left_win.m; it is used by depot.m at selection. It defines the pair-wise comparison of the trial and target vectors for input into the next generation. The package is developed further with code specific to the Penalty Cost Model. OBjfun.m has a number of files required to implement the Penalty Cost Mode. Equation (2.4) is implemented in expon.m while equation (2.2) is calculated using backlog.m. Extra modifications to the code were necessary to implement the following subsections, 4.3.1, 4.3.2 and 4.3.3.

4.3.1 Boundary Constraints

Rundeopt.m required significant adjustments to test various DE testing scenarios. The test bed of 90 scenarios is supplied by CSV file. For each scenario, DE is required to create bounds for an initial population of vectors. Obviously, the optimum solution should be inside this bounded region. Typically, the bounds are a function of the number of parameters in the objective function. However, a more robust method of bounds calculation is required for the Penalty Cost Model to guarantee the optimum solution is covered by the initial population region. A flexible method was developed using the input information from each scenario. The CW lead-time (L_0), FSL lead-time (L_i), FSL arrival rate (λ_i), unit cost (c) and penalty cost (p) are used to create the upper bound for a particular FSL *i*.

Comparisons of the level at which the cumulative probability of the Poisson distribution with mean

$$a = (L_i + L_0) * \lambda_i \quad \textbf{(4.1)}$$

which traverses the level of (4.2) for the first time is compared to (4.3).

$$p/(p+c) \quad \textbf{(4.2)}$$

$$u = a + 3 * \sqrt{a} \quad \textbf{(4.3)}$$

The greater value is used as the upper bound for the FSL. In other words,

$$UB_i = \max\left\{\min\left\{x \mid \Pr\{\omega \le x\} \ge \frac{p}{p+c}\right\}, a + 3*\sqrt{a}\right\}, \quad \textbf{(4.4)}$$

where UB_i is defined as the upper bound for decision variable S_i, and ω is a Poisson random variable with mean a. This is repeated until upper bounds are created for all FSLs: $UB_1,...,UB_J$. A summation of the FSL upper bounds is assigned to S_0 as an upper bound:

$$UB_0 = \sum_{i=1}^{J} UB_i, \quad \textbf{(4.5)}$$

4.3.2 Discrete values

Another adaptation to the code was necessary to mitigate DE's continuous nature. DE is a meta-heuristic algorithm for global optimisation over continuous spaces. The Penalty Cost model is an objective function with continuous and discrete input variables. The discrete input variables, which are also the decision variables, are stock levels across the CW and FSLs: $S_0, S_1,..., S_J$. The initial population and subsequent iterative generations of new populations in DE hold vectors of input variables (stock levels) which are consumed by the objective function as the algorithm converges towards the optimum solution. As a result, it is necessary to adjust all vector populations to discrete values. In effect, this turns the implementation of DE into discrete DE.

During the initial population stage, DE will generate a matrix of real numbered vectors. As discrete values are required, a method of translating the matrix of real values into a matrix of integer equivalents is required. Simply rounding to the nearest Integer is insufficient. Due to the nature of the services supply chain business, a difference of one unit can have implications to the total cost outcome. Rounding to the nearest integer in theory could provide an optimum solution that is lower then the actual optimum solution provided by the best-in-class method. Taking this into consideration, a random number process of rounding up or down is used to seed the initial population and subsequent generations with discrete values. This random process is used again during the algorithm's boundary constraints management stage to ensure that mutated values are inside the upper and lower bounds and is also discrete in nature.

4.3.3 Alteration of initial population

After an exploratory review of the results using standard DE, certain scenarios were identified that failed to provide the optimum solution. They were consistently shown to deviate from the best-in-class algorithm with significant max and mean deviations. It was known from the best-in-class method that the optimum solution for the scenarios is a solution where no stock is required to be placed at the central warehouse or FSLs. It was decided to introduce a vector representing this solution into the initial population. The initial population is a random generation process so it possible that a vector of

zeros could be created. The lower and upper bounds are used to create a population of vectors that are possible solutions. Hence, the initial population could contain a vector matching the solution outlined above. However, it is not guaranteed and with respect to the scenarios that struggled, it was a requirement. A method of calculating distance from zero is applied to each vector in the initial population to select one for replacement. The distance between vector [a,b,c,d] and [0,0,0,0] is defined as

$$Dist = a\text{^}2 + b\text{^}2 + c\text{^}2 + d\text{^}2 \quad \textbf{(4.6)}$$

The vector with the closest distance to [0,0,0,0] is replaced by a vector of [0,0,0,0] . As shown in the numerical results, this improves DEs performance in achieving the optimum solutions.

Chapter 5

5 Experimental Setup and Numerical Results

5.1 Introduction

This chapter presents the results of a comparison between a DE implementation of the Penalty Cost Model against a best-in-class method developed in the field of Operations Research. The purpose of the analysis is to test the performance of DE in a variety of different scenarios. The results are introduced for two DE algorithm variants, algorithm two and three are presented in the order they were compiled. They correspond to equations (3.7) and (3.8) which are DE/local-to-best/1/bin and DE/best/1/bin with jitter in respective DE notation. Both algorithms apply the unmodified standard DE and the enhanced DE code changes described in chapter 4, sections 4.3.1 to 4.3.3. A comparison of standard DE (unmodified) and enhanced DE to the best-in-class method is presented for each algorithm followed by a comparison of each DE algorithm to ascertain the better performing algorithm in the context of the Penalty Cost Model.

The numerical results chapter is structured as follows;

- Test-bed design.
- Control and experimental setup.
- Review of the DE control parameters.
- Performance analysis of the DE.
    - Algorithm three
    - Algorithm two
    - Algorithm comparison
- Scalability of the DE Penalty Cost Method

5.2 Test-Set Design

As mentioned in Chapter 4, a full factorial test set of 90 scenarios was designed to test the ability of DE to converge to the optimum solution. Three levels of values, low, medium, high are used for the input parameters. Each scenario contains the following input data; unit cost (c), penalty cost (p), central warehouse (CW) lead times (L_0), Forward Stocking location (FSL) lead times (L_i), and FSL arrival rate (λ_i). The FSL arrival rate is the rate in which the FSL will receive requests for replacement units

from the customer per calendar day. The low, medium, high arrival rates values are 0.01, 0.1 and 1, respectively. An arrival rate of one is considered high in the context of service parts as this would equate to 30 requests for replacement units per month.

The unit cost is fixed at 1 for all 90 scenarios, and the penalty cost is varied using the values 9, 999 and 99999 for low, medium and high. The central warehouse lead times are one, five and ten calendar days. This lead time is the length of time to repair and return repaired units to the CW. The FSL lead times are all fixed at 1 day ($L_i = 1$), and the number of FSLs is restricted for 3 ($J = 3$). A full factorial design yields 90 distinct scenarios (Three penalty cost values * three CW lead time values * 15 different arrival rate combinations for three FSLs). A sample set of input parameters for five scenarios is given in Table 5.1. All 90 scenarios are designed to test the performance in different parameter regimes. High values of penalty cost coupled with a low CW lead-time and high arrival rates at the FSLs will produce an optimum solution in a different region to a scenario with low penalty cost, and arrival rates, and high central warehouse lead time.

Table 5-1 Sample input scenarios

Scenario	c	p	L_0	L_1	λ_1	L_2	λ_2	L_3	λ_3
1	1	9	1	1	0.01	1	0.1	1	1
2	1	9	5	1	0.1	1	1	1	0.01
3	1	999	1	1	0.01	1	0.1	1	1
4	1	999	10	1	1	1	0.01	1	0.1
5	1	99999	1	1	0.01	1	0.1	1	1

5.2.1 Control and Experimental Setup

The code used in this study is the DeMat Matlab package [1]. A modified version of the code is used to process the input file containing 90 scenarios. A scenario defines a particular set of problem parameters. The input is supplied by CSV file in the form of Table 5.1. DE was calibrated to select the best set of control parameters from an iterative review of all DE control parameters. The selected control parameters are hard-coded as part of the control setup. A review of the selection process is discussed later in the chapter. Maximum number of generations for one scenario is set at 500. Results are aggregated in CSV files. Incorporated in the results are a mixture of dependent and independent variables. The total cost calculated by the objective function is observed along with the optimum solution of stock levels for CW and FSLs. The iteration that DE converged on the optimum solution is collected and the computation time for 500 iterations. The hardware and software used for testing is a Dell PowerEdge 2900 server, two 64-bit Dual-Core Intel® Xeon® Processor 3.20 GHz, 1M Cache, 800 MHz with Redhat and uncompiled Linux version of Matlab.

Table 5-2 Sample results

Scenario	F	CR	Total Cost	CW	FSL 1	FSL 2	FSL 3	Iteration	Comp Time
1	0.5	0.9	0.27	0	0	0	0	1	9.9558
2	0.5	0.9	15.396	3	3	3	3	3	12.812
3	0.5	0.9	10.433	1	4	4	0	1	17.525
4	0.5	0.9	29.883	13	2	4	10	6	43.599
5	0.5	0.9	47.933	23	10	10	4	12	63.173
6	0.5	0.9	45.845	22	10	10	3	4	61.183

Note: F is the value that is used in mutation to create the scaled difference vector. CR is the value most commonly used in binomial crossover. It determines how much of the parent vector is provided along with the mutant to create the trial vector.

Using scenario five as an example, the recommended stock levels at the locations is displayed in columns CW, FSL1, FSL2, FSL3, and the minimum investment cost based on the various inputs supplied to this scenario is 47.933. This recommendation will minimise the total investment to negate the possibility of any penalty costs applied. As part of the control setup, an examination of DE's performance against the 90 scenarios is required. A known best-in-class method was developed in conjunction with researchers at Bell Labs to produce the optimum solution using the same 90 scenarios. It is a gradient descent search algorithm using METRIC approximation [2] to calculate the variable W_0 and is described in more detail in equation 2.4. The method is referred to as the C Penalty Cost Algorithm (best-in-class) as the algorithm was developed in C. Results similar to Table 5.2 are produced in CSV format. Comparison between both methods is validated using mean percentage difference and a visual assessment of the optimum solutions for both methods. This assessment gave rise to the modifications outlined in chapter 4, section 4.3.3. To distinguish the modified version of DE from the original, the methods are referred to as Standard DE and Enhanced DE. Both methods apply the same DE control parameters.

5.3 Review of DE Control Parameters

There are a number of control parameters that are fixed for each DE generation. An iterative review of the control parameters was taken to select a combination for the Penalty Cost Model. The three control parameters adjusted are the step size otherwise known as the scale factor F, the crossover probability CR and the DE algorithm variant. There are six DE variants as part of the DeMat Matlab package. Each variant applies different mutation techniques to converge on the optimum solution. The following range of parameters for scale factor F was used, {0.5, 0.6, 0.7, 0.8, 0.9, 1, 1.1}. The range advised in the literature is [0,2] although it is recommended to stick between the region [0.5 1].

The CR control parameter has a range of [0 1]. The values used in the review are 0.1, 0.3, 0.5, 0.7, 0.8, 0.9 and 1. Based on recommendations in the Matlab package, it was decided to focus on the region 0.7

to 1 and this is why the incremental difference is 0.1. We still consider CR probabilities outside this and provide results for the region 0.1 to 0.7 in 0.2 incremental jumps. The number of scenarios used for the iterative review of control parameters is 27 with a similar spread of problem settings to the test-bed of 90 scenarios. The number of generations within DE is set to 1000. For each combination of control parameters, a total of 294 separate iterations are collected for each scenario. The collection of results is provided in Figure 5.1 displaying each DE variant's performance as a function of the scale factor F and the crossover probability CR, with the coloured cells representing the log scale of the number of iterations to converge at the optimum solution.

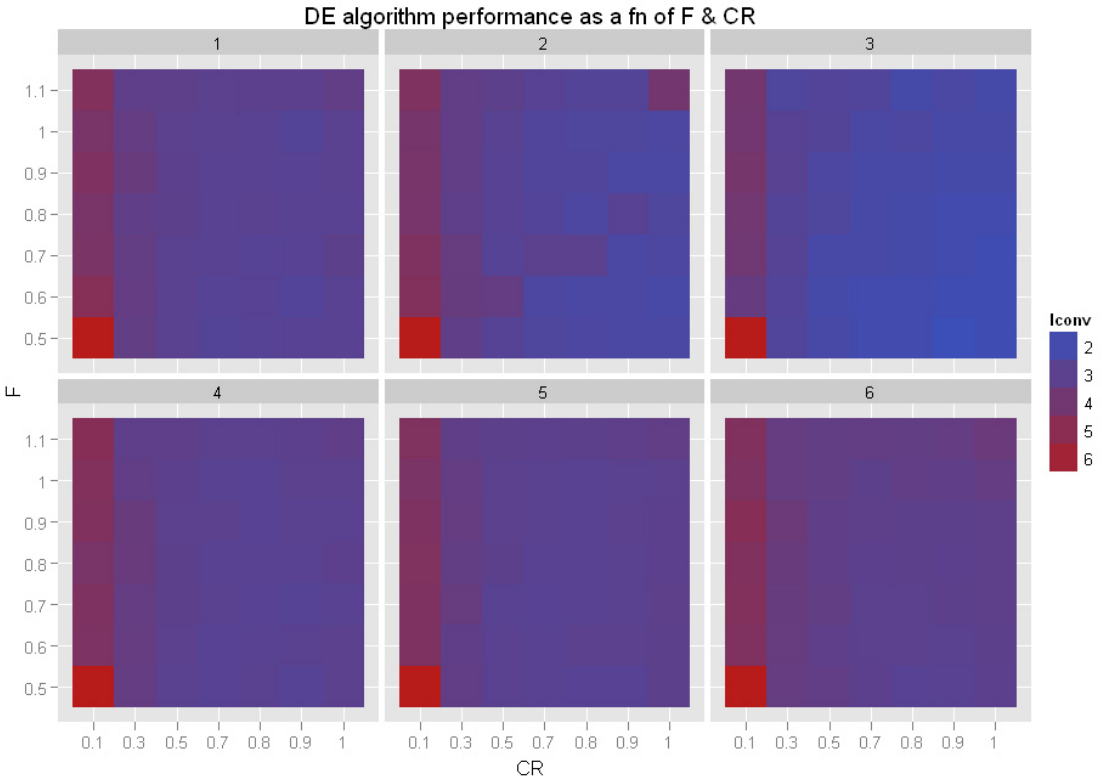

Figure 5.1 DE algorithm performance as a function of F & CR

Note: Best combination based on convergence to optimum is F = 0.5, CR = 0.9 and DE algorithm 3.

The best combination of parameters suggested by this review is F = 0.5, CR = 0.9 and algorithm three. Figure 5.11 Algorithm two suggests an alternative choice holding F and CR to the same values as the best combination. Due to the close call, both algorithms two and three are used in the numerical experiments and the same F and CR values are used in the testing process.

5.4 DE Performance Analysis

The DE control parameters used for testing are two variants of the DE algorithm, $C_R = 0.9$ and $F = 0.5$. The test-bed of 90 scenarios is collated 30 times for each DE algorithm and compared to the best-in-class algorithm. Results are supplied for the unmodified standard DE and the enhanced version of DE.

To determine the accuracy of the solutions returned, percentage deviation from the cost of the best-in-class algorithm is used as a performance metric which is defined as;

$$PD = \frac{D_{Cost} - C_{Cost}}{C_{cost} * 100}, \quad \textbf{(5.1)}$$

where C_{Cost} is the total cost of the best-in-class algorithm and D_{Cost} is the total cost supplied by DE for a particular scenario. Each scenario is run 30 times to identify the stability of DE. Labels such as acceptable, not acceptable, or requires further analysis are used to describe the performance of DE in each scenario. Obviously, scenarios that achieve the best-in-class optimum are acceptable. Scenarios with a max deviation less than or equal to 2% over 30 iterations and an overall mean deviation of no more that 1% over 30 iterations is also considered acceptable. Scenarios outside of these regions are either unacceptable or require further analysis. Unacceptable scenarios have a max deviation more than 2% and an overall mean deviation more than 1%. Scenarios that require further analysis are in the grey area of acceptability, and need to be judged on a case by case basis. In general, the performance is good but deviated significantly at certain iterations. Whether scenarios are placed into the acceptable area depends on the tolerance of the decision maker. Almost all supply chain systems contains multiple items with different unit costs. As a result, the focus will be on the more costly items in the system where the tolerance will be tightened as apposed to a relaxation of the tolerance for inexpensive items. Table 5.3 displays a measure of a scenario's stability for DE algorithm three.

5.4.1 Algorithm 3 - DE/best/1/bin with jitter

For 30 iterations, standard DE achieves stable solutions in 68 scenarios achieving the best-in-class optimum in 49 cases. It returns seven scenarios that are unacceptable. The maximum percentage deviation is 25.14. The average deviation over the runs that yield positive deviation is 4.31. Note that this measure gives the expected percentage deviation in case there is a deviation from the best-in-class solution. For a real world Penalty Cost Model with overall investment in millions of dollars, the deviations figures for standard DE may be considered high. This is in contrast to enhanced DE that does not return any unacceptable scenarios. It achieves the optimum in 56 cases with an overall acceptable figure of 75 out of 90 scenarios with a maximum percentage deviation of 8.73, and an average deviation (over the runs that yield positive deviation) of 2.24 respectively. Note the considerable decrease in both maximum and average percentage deviation figures. Although there is significant improvement in terms of results, enhanced DE is not dominant over standard DE for algorithm three. There are scenarios in which standard DE achieves the optimum when enhanced DE fails, and vice versa. On the one hand, there are six scenarios for which Standard DE achieves the solution of the best-in-class algorithm while enhanced DE does not. On the other hand, Enhanced DE achieves the solution of the best-in-class algorithm in 14 scenarios where Standard DE yields positive percentage deviations. There is an overlap of 24 scenarios where both standard and enhanced DE fails to achieve the best-in-class optimum although 11 of the scenarios are considered acceptable. Further

analyses of the problem settings for the overlapping scenarios do not identify a pattern that would account for the persistent deviations.

Table 5-3 DE Algorithm 3

Scheme	Standard DE	Enhanced DE
Achieved Optimum	49	56
max dev <= 2, mean dev <= 1 (acceptable)	19	19
max dev > 2, mean dev > 1 (unacceptable)	7	0
max dev > 2, mean dev <= 1 (grey)	15	15

Note: Standard DE has 7 un-acceptable scenarios

5.4.2 Algorithm 2 - DE/local-to-best/1/bin

For 30 iterations, standard DE achieves stable solutions in 78 scenarios achieving the best-in-class optimum in 72 cases. It returns 12 scenarios that are unacceptable. The maximum and average (over the runs that yield positive deviation) percentage deviation is 69.17 and 16.50 respectively. Deviations of this magnitude are impractical for use in any supply chain system. The deviations are concentrated in a group of scenarios that are consistent with the unacceptable scenarios from algorithm three. This is the purpose of the modifications to DE outlined in chapter 4. Contrasting these figures with enhanced DE for algorithm two, it achieves the optimum in 80 cases with an overall acceptable figure of 88 out of 90 scenarios. It has a maximum and average (over the runs that yield positive deviation) percentage deviation of 4.01 and 1.15 respectively. Enhanced DE shows significant improvements over standard DE in the amount of optimum solutions achieved and also in the exceptional reduction in max and average deviations. Similar to algorithm three, enhanced DE is not dominant over standard DE although there is a noticeable shift in partial dominance towards enhanced DE. There are three scenarios in which standard DE achieves the optimum when enhanced DE fails. However, enhanced DE achieves the solution of the best-in-class algorithm in 12 out of 14 scenarios where Standard DE yields positive percentage deviations.

Table 5-4 DE Algorithm 2

Scheme	Standard DE	Enhanced DE
Achieved Optimum	72	80
max dev <= 2, mean dev <= 1 (acceptable)	6	8
max dev > 2, mean dev > 1 (unacceptable)	12	0
max dev > 2, mean dev <= 1 (grey)	0	2

5.4.3 Problem Settings

The numerical results identify a number of scenarios where DE has failed to achieve the optimum. Table 5.5 identifies the scenarios that algorithm three failed to achieve the optimum for both standard and enhanced DE. Table 5.6 is the corresponding scenarios for algorithm two. The purpose of identifying the scenarios is to check if there is an underline pattern to the scenarios which DE fails to achieve the optimum. There is no clear trend other then medium to high penalty costs and medium to high CW lead time. Both variables are used in the creation of an initial population of vectors and provide more opportunities for the DE algorithm to become trapped in local minima. DE does not guarantee that the optimum solution will be found. It is the nature of stochastic algorithms converging to the global optimum.

Table 5-5 Scenario overlap – Standard and Enhanced DE

Scenario	c	p	L_0	λ_1	λ_2	λ_3
32	1	999	1	0.01	0.01	0.1
34	1	999	1	0.01	0.1	0.1
43	1	999	5	0.01	0.01	1
55	1	999	10	0.1	0.1	0.1
80	1	99999	5	1	1	0.01
83	1	99999	10	0.01	0.01	1
87	1	99999	10	0.01	0.1	1

Table 5-6 Algorithm 2 Scenario overlap – Standard and Enhanced DE

Scenario	c	p	L_0	λ_1	λ_2	λ_3
8	1	999	1	1	1	1
32	1	999	1	0.01	0.01	0.1
45	1	999	5	0.1	0.1	0.1
46	1	999	5	1	0.1	0.1
58	1	99999	10	1	1	1

5.4.4 DE Algorithm Comparison

Providing an effective method for comparison between both DE algorithms, Algorithm 2 and 3, allows for the formulation of specific instances where one would favour a particular algorithm over the other. For the Penalty Cost Model, it is shown that the standard DE method could not be applied to any real world system due to the number of unacceptable solutions and the high percentage deviations returned. Concentrating on enhanced DE as a way of comparing Algorithm 2 and 3, it demonstrates that algorithm two is more accurate and comparable to the best-in-class method due to the number of optimum solutions and the low overall percentage deviations.

We see an increase in the number of optimum solutions attained by Algorithm 2 and a reduction in max and mean percentage deviations as shown in Table 5.7. The increase in optimum solutions allows Algorithm 2 to hold dominance over Algorithm 3. Algorithm 2 achieves the optimum best-in-class solution in 23 scenarios where Algorithm 32 yields positive percentage deviations. Nonetheless, for particular scenarios this accuracy comes at the expense of computation time and the number of iterations to converge. As a result, scalability is a factor and leads to certain conclusions about each algorithm in relation to the model and supply chain distributed systems in general.

Table 5-7 MAX & AVERAGE Percentage deviations Algorithm 2, Algorithm 3

% Deviation	Standard DE		Enhanced DE	
Scenario	MAX	AVERAGE	MAX	AVERAGE
Algorithm 3	25.14	4.31	8.73	2.25
Algorithm 2	69.17	16.50	4.01	1.15

Note: The average percentage deviation is calculated over the runs that yield positive deviation, not all runs.

For a system with larger numbers of forward stocking locations combined with high levels of penalty cost, Algorithm 2 could be preferred for its accuracy in determining the optimum solutions. An extra consideration of inexpensive items signifies a relaxation in the acceptable tolerance of the system which counteracts the lower probability of finding the optimum solution. In this case, we may be happy to use Algorithm 3 to take advantage of its convergence rate. In the next section, we discuss whether using Algorithm 2 for particular scenarios is computationally too expensive to the faster converging Algorithm 3.

5.5 Scalability Analysis

The initial experimental setup used a full factorial test bed of 21 scenarios. The scalability review is applied to both algorithms using the enhanced DE method. The CW lead-time is constant throughout and set to high (10). The Penalty Cost is adjusted from low to medium to high. The number of FSLs is 3, 6, 12, 24, 48 and 60. The FSL lead-time is constant as it is the case in the initial 90 scenario test-bed. The FSL arrival rate for each scenario is a randomly generated real number between [0,1]. As the underlying optimisation problem, the Penalty Cost Model, is combinatorial in nature, and DE is a computational algorithm, it is expected that the computation time will increase exponentially as the size of the population grows. Figure 5.2 shows the minute differences in the computation time of each algorithm for penalty cost, 999 but in general, the computation times are similar.

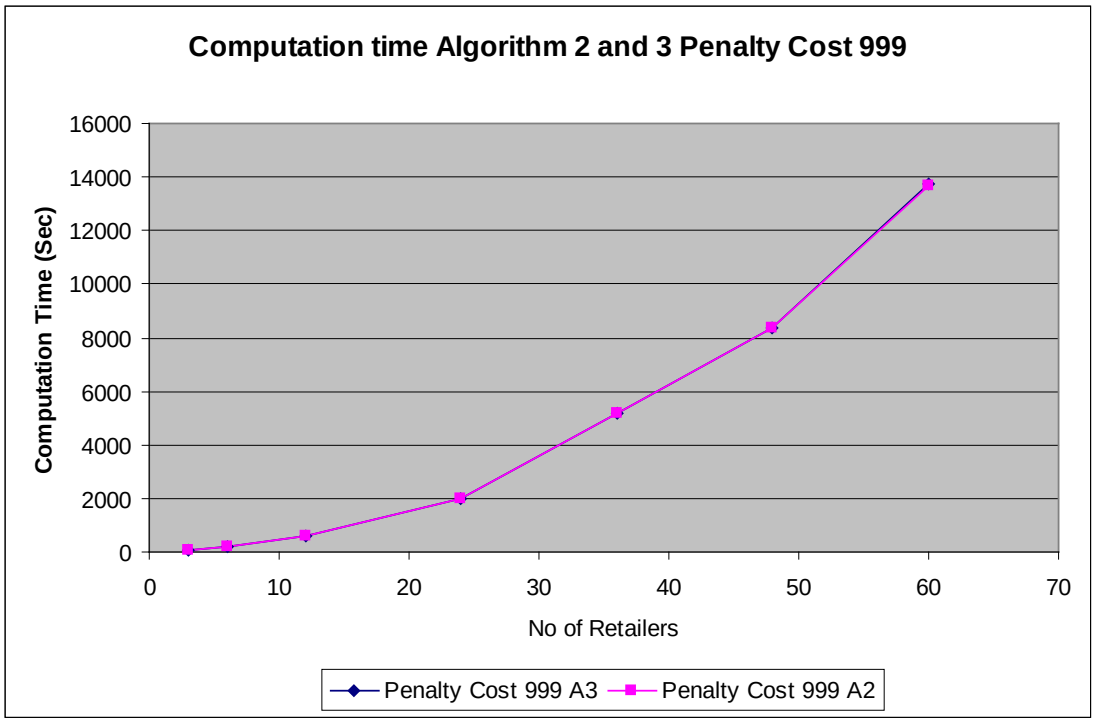

Figure 5.2 Computation time for Algorithm 3 and 2 at penalty cost 999

The number of iterations to convergence is more intuitive and provides good insight into the choice of algorithm and under what conditions that choice is advised. Figure 5.3 shows the iterations as a function of the number of FSLs at each penalty cost with a mean value obtained. As the figure shows clearly, the number of iterations depends heavily on the value of the penalty cost, which is intuitive. Scenarios with high penalty costs have larger upper bounds (see equation 4.4) which grow the search space significantly.

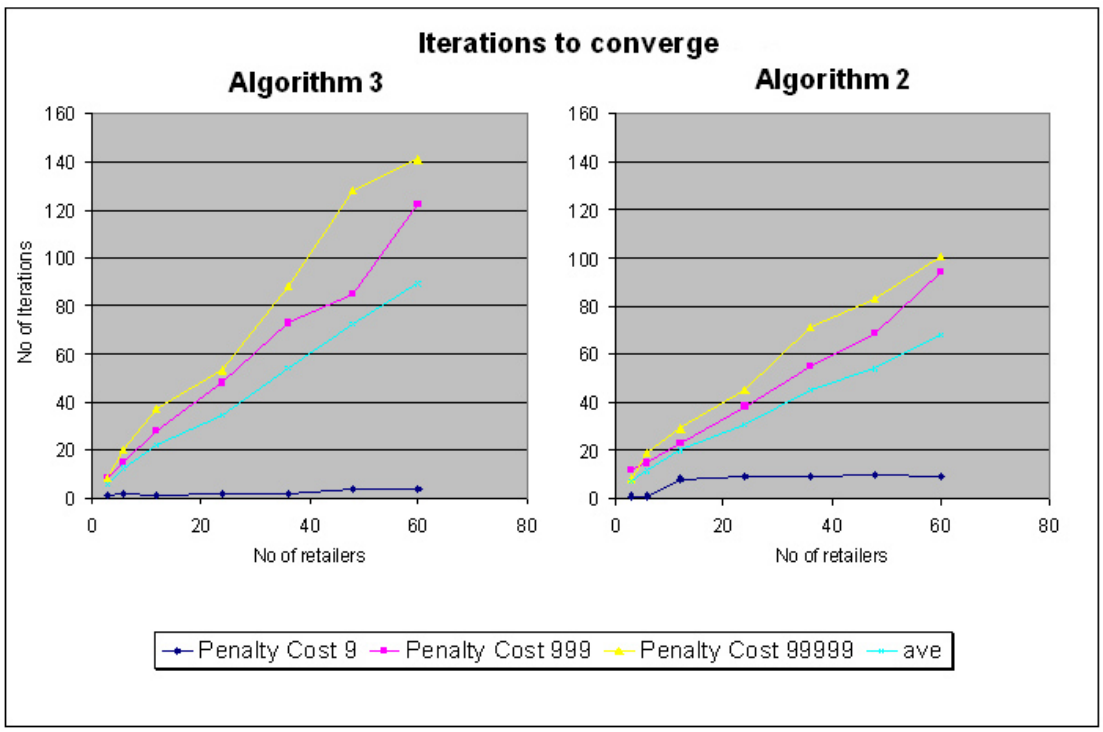

Figure 5.3 Algorithm 3 and Algorithm 2 Iterations to convergence

Figure 5.4 to Figure 5.6 show the different convergence rates of each algorithm at penalty cost 9, 999, 99999 respectively. We see that for small number of retailers which are analogous to smaller population sizes, algorithm three converges on the optimum faster. In relation to the test-bed, scenarios with penalty cost 9 have smaller populations then the rest of the test-bed. As the penalty costs increase, the boundary constraint modifications discussed in 4.3.1 ensure that the population size increases accordingly. In addition, the scalability test-bed differs from the original 90 scenario test-bed in that the number of retailers increases for every scenario.

This drastically changes the population size for each scenario in the scalability test-bed. As the population size increases with medium to high levels of penalty cost, algorithm two performs better at converging to the optimum. In light of this and taking the accuracy of solutions into consideration, algorithm two is the suggested algorithm to use for penalty cost models with a number of forward stocking locations greater than six and penalty costs in the medium to high range.

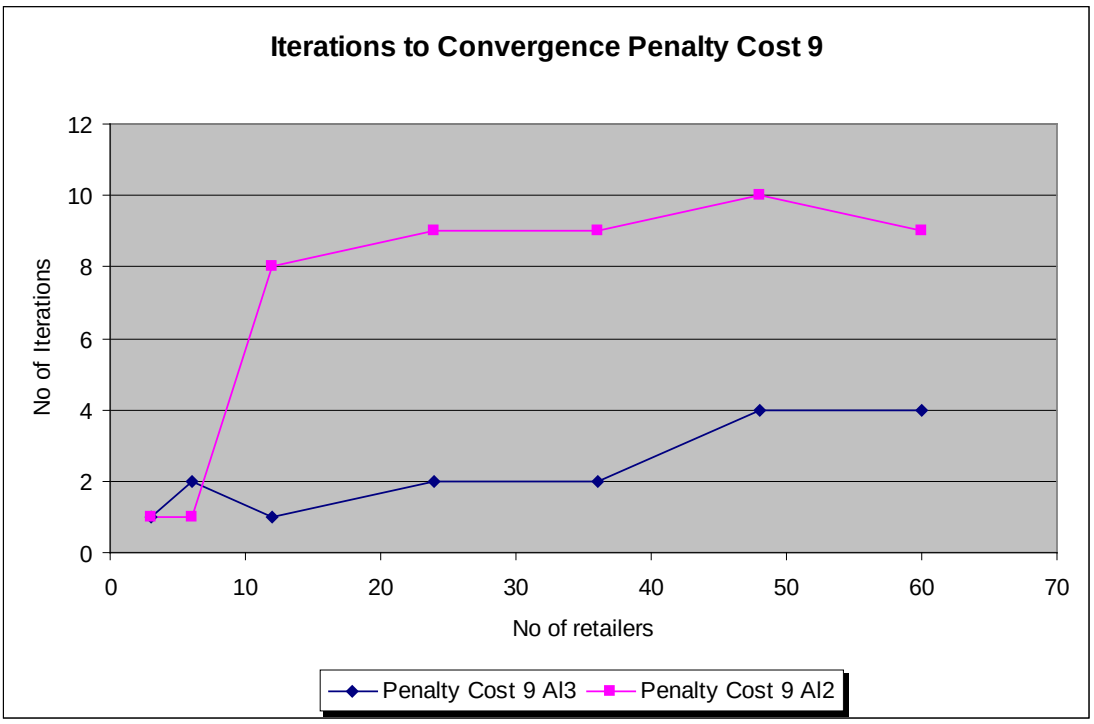

Figure 5.4 Penalty Cost 9 Algorithm 2 & 3 Iterations to converge

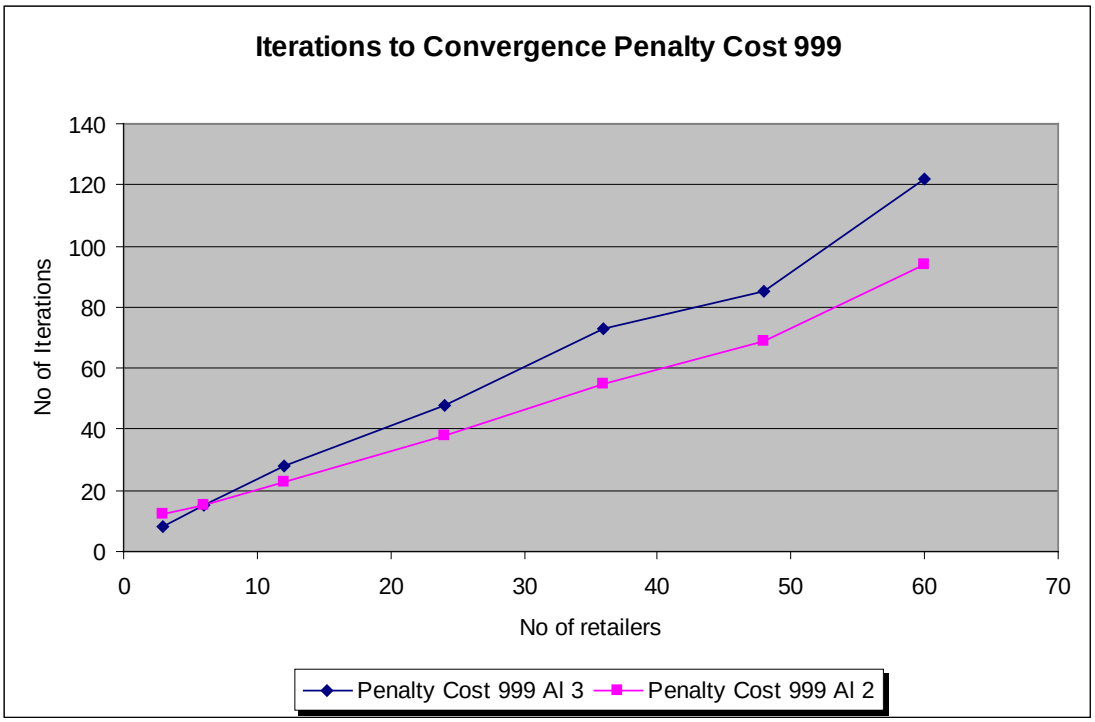

Figure 5.5 Penalty Cost 999 Algorithm 2 & 3 Iterations to converge

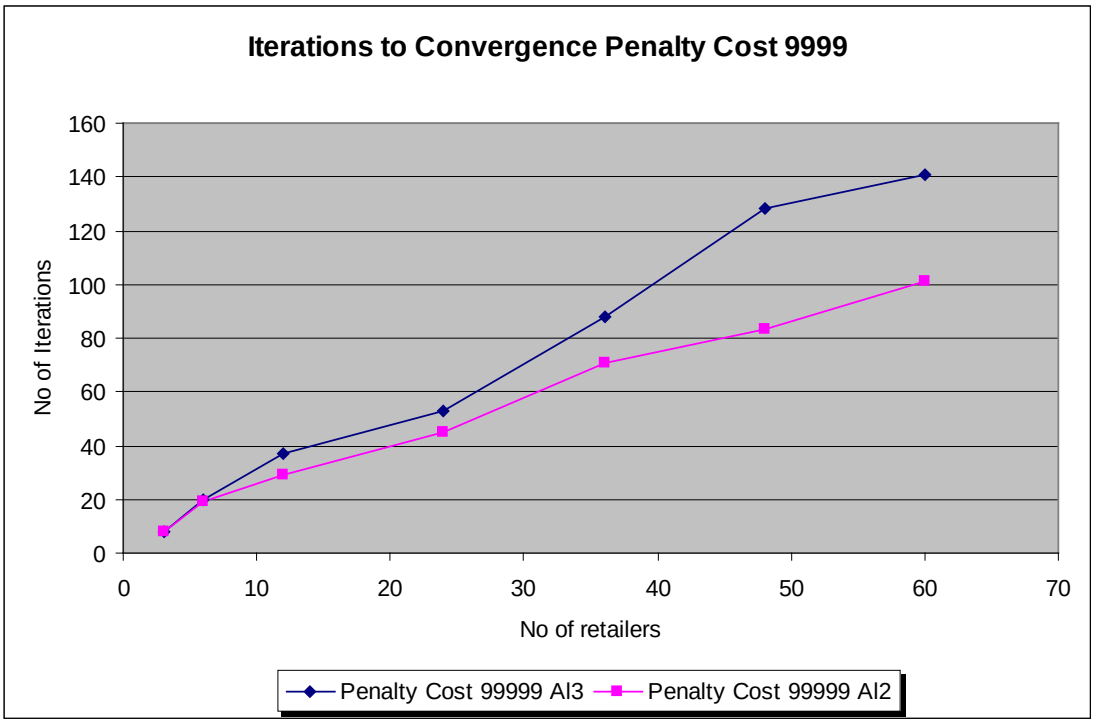

Figure 5.6 Penalty Cost 99999 Algorithm 2 & 3 Iterations to converge

Chapter 6

6 Conclusions and Further Research

The two echelon inventory system for spare parts is a well-known model with real world applications in inventory management, manufacturing and production planning. In most if not all large technology corporations, there is a plentiful supply of optimisation problems that can be addressed with inventory planning and cost reductions at the forefront. Up until now, there have been sporadic reviews and applications of supply chain system optimisation using DE. In this thesis, the goal was the application of Differential Evolution to the optimisation of spare part inventory in a two-echelon supply chain setting. The main motivation of this thesis was to gain an understanding of the mechanics of Differential Evolution and to determine the accuracy of DE in generating the optimum solutions for the Penalty Cost Model, which is the underlying optimisation problem for the aforementioned inventory system.

6.1 Aims and Objectives

The Penalty Cost Model is the mathematical optimisation model for determining the inventory requirements (where to stock and how much to stock) in a two-echelon spare parts supply chain system consisting of one central warehouse and a number of forward stocking locations. Customer requests for spare parts need to be fulfilled from on-hand stock, and any backlog results in penalties charged by the customer as part of the service contract between the parties. The objective of the Penalty Cost Model is to minimise the expected costs of the system that are comprised of the item costs (investment in inventory) and expected penalty costs. Based on preliminary results, a number of code modifications where introduced into the DE package. The modifications relate to boundary settings and initial population generation which are internal functions to DE. As the complexity of the model intensifies, so does the need to ascertain the scalability of DE. Hence, we have conducted a numerical experiment to test how DE scales as the size of the problem grows, which is achieved by increasing the number of forward stocking locations. The performance of each DE algorithm is assessed at ever increasing forward stocking location numbers and conclusions as to which algorithm is the preferred choice under complex Penalty Cost Model conditions.

6.2 Outcomes

We applied the DE algorithm to a well-known and very hard discrete non-linear stochastic optimisation problem used to determine the inventory investment in a two-echelon spare parts inventory system. This is important because the use of DE in supply chain planning is limited, and the underlying optimisation problem is notoriously difficult since it has discrete decision variables and is combinatorial in nature. An initial step was to conduct a small numerical study to decide on which DE variant and control parameters to use. This study showed that Algorithm 3 (DE/best/1/bin with jitter) and Algorithm 2 (DE/local-to-best/1/bin) outperformed the others, and the parameter values to use for the supply chain problem are F scale factor, 0.9 and cross over (CR) probability, 0.5. The main experimental setup and numerical results were motivated by real life supply chains where the various Penalty Cost Model parameters are varied to induce more complex Penalty Cost scenarios in which DE must achieve the optimum. The varied parameters are the penalty cost, the central warehouse lead time and the arrival rates at the forward stocking locations. A full factorial design of a three-level set up resulted in a test bed of 90 Penalty Cost Model scenarios. Using this test-bed, the performance of Algorithm 2 and Algorithm 3 was tested. Using a metric referred to as percentage deviation, each DE algorithm was compared to the best-in-class analytical algorithm in the field of Operations Research. Note, percentage deviation (Equation 5.1) is the percentage difference between the best-in-class cost and the DE cost and the test-bed of 90 scenarios was run 30 times for each algorithm. The percentage deviations for Algorithm 2 were found to have a maximum and average (over the runs that field positive deviation) of 69.17% and 16.50% respectively. Algorithm 3 showed improvements with a maximum and average (over the runs that field positive deviation) of 25.14% and 4.31% respectively. We found that neither Algorithm 2 nor Algorithm 3 were appropriate for real like applications since the solutions found by the algorithms deviate considerably in many cases. Consequently, we decided to modify the standard version of each DE algorithm to introduce enhancements for a more robust means of boundary calculation, to facilitate the Penalty Cost Model's discrete decision variables and to adjust the initial population of solutions generated by DE before optimisation. We call the modified algorithm enhanced Algorithm 2 and enhanced Algorithm 3. The numerical results using the same test bed for the same number of iterations found the performance to increase considerably. The maximum and average (over the runs that field positive deviation) for Algorithm 2 were 4.01% and 1.15% respectively. The corresponding maximum and average for Algorithm 3 were 8.73% and 2.25% respectively. The culmination of the numerical study was to test the scalability of the enhanced DE algorithms and it was found that Algorithm 2 is the preferred algorithm for optimising the Penalty Cost Model at all levels of scenario complexity. Our figures show that for increasing population size, Algorithm 2 outperforms Algorithm 3 using the number of iterations to convergence as the indicator We are confident that as a standalone application, the enhanced Algorithm 3 under certain real life supply chains will produce Penalty Cost Model solutions that are considered acceptable. However, enhanced Algorithm 2 outlines a more consistent stable application with optimum solutions of the best-in-class achieved in 97% of the cases. It returns the best-in-class solution in the bulk of scenarios where Algorithm 3 presents positive percentage deviations. In the context of achieving the best-in-

class solution, the accuracy of Algorithm 2 signifies that it is the recommended algorithm to adopt for the Penalty Cost Model. In light of the results of this research we provide the following conclusions:

1. Is DE an effective optimisation tool for solving the Penalty Cost Model?
   No, the standard version of DE was not effective at achieving the best-in-class solutions and the percentage deviations outlined present an unacceptable level of performance. However, the enhanced version of DE is and the recommended DE algorithm to use is Algorithm 2.

2. How does it compare with the best-in-class analytic algorithm?
   Enhanced DE Algorithm 2 achieves the optimum in 97% of the scenarios and the deviations in the scenarios where it returned positive percentage deviations are well inside real life supply chain tolerances. In the context of optimum solutions achieved and the nature of stochastic algorithms like DE where no guarantee of convergence is given, we show that the enhanced DE Algorithm 2 is comparable to the best-in-class analytical algorithm. In terms of practical applications of the enhanced DE Algorithm 2, the computational time increases exponentially for larger population sizes. See Figure 5.3 for the computation time for Algorithm 2 and Algorithm 3 at penalty cost 999. In real life supply chain settings, a distribution of 24 forward stocking locations is considered a large collection of local warehouses to optimise. In consideration, Algorithm 2 converges on the optimum in 4.37 minutes using uncompiled Matlab on a Dell PowerEdge 2900 server, with a 64-bit Dual-Core Intel® Xeon® 3.20 GHz Processor.

3. Is the DE implementation effective at achieving the optimum solutions for a given test-bed?
   Using the standard version of DE for both algorithms, we show that it is not effective at achieving the optimum solutions. Applying the enhanced version of DE, Algorithm 2 is effective at achieving the optimum solution under all Penalty Cost Model conditions. Algorithm 3 is effective at only certain Penalty Cost Model conditions.

4. Under what conditions does DE fail to achieve the optimum solution?
   In light of the fact that Algorithm 3 is successful only at certain Penalty Cost Model conditions, we know that for larger populations, it does not perform as well as Algorithm 2. The numerical results supplied by Algorithm 3 under these specific conditions are acceptable but not as accurate as the results for Algorithm 2. Larger populations' contain decision variables where the penalty cost, the central warehouse lead times, and the number of forward stocking locations are increasing. In the context of the Penalty Cost Model scenarios, Algorithm 3 is comparable to Algorithm 2 only at smaller levels of each decision variable.

6.3 Further Research

The penalty cost objective function optimised by DE presents some interesting avenues of research. The study can be applied to significantly more complex supply chain objective functions that are considered difficult to optimise in the field of Operations Research. The scalability assessment of the current DE implementation could be explored utilising DE's parallel processing capabilities at higher levels of complex Penalty Cost Model conditions. Applying parallel processing to this Penalty Cost Model would greatly reduce the computation time and may allow a practical DE implementation of the Penalty Cost Model. The advantages of using this would appear at much larger number of retailers as the search space is more extensive.

References

[1] PRICE, K., STORN, R., LAMPINEN, J 2005. Differential Evolution: – A Practical Approach to Global Optimization. Springer, Heidelberg (2005)

[2] SHERBROOKE, C.C 1968. METRIC: A Multi-Echelon Technique for Recoverable Item Control. Operations Research (5.2.1)

[3] LAMPINEN, J 1999, A Bibliography of Differential Evolution Algorithm, Technical Report, Lappeenranta University of Technology, Department of Information Technology, Laboratory of Information Processing.

[4] STORN, R, PRICE, K. 1995 ICSI technical report Differential Evolution – A simple and efficient adaptive scheme for global optimisation over continuous spaces

[5] STORN, R, PRICE, K. 1996: Minimizing the real functions of the ICEC contest by differential evolution. In: Proceedings of the 1996 IEEE international conference on evolutionary computation, Nagoya, Japan, pp. 842–844. IEEE Press, New York (1996)

[6] STORN, R. 1996: On the usage of differential evolution for function optimization. In: Smith, M.H., Lee, M.A., Keller, J., Yen, J. (eds.) Proceedings of the 1996 biennial conference of the North American fuzzy information processing society – NAFIPS, Berkeley, CA, USA, June 19–22, pp. 519–523. IEEE Press, New York (1996)

[7] STORN, R, PRICE, K. V, 1997.: Differential evolution: a simple evolution strategy for fast optimization. Dr. Dobb's Journal 22, 18–24

[8] STORN, R, PRICE, K.V. 1997: Differential Evolution – a Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. Journal of Global Optimization 11(4), 341–359 (1997)

[9] ZAHARIE, D. 2002: Critical values for the control parameters of differential evolution algorithms. In: Matoušek, R., Ošmera, P. (eds.) Proceedings of MENDEL 2002, 8th international conference on soft computing, Brno, Czech Republic. Brno University of Technology, Faculty of Mechanical Engineering, June 5–7, pp. 62–67. Institute of Automation and Computer Science, Brno

[10] HOOKE R & JEEVES T A 1961. "Direct search" solution of numerical and statistical. Research Development Center. Westinghouse Electric

[11] KIRKPATRICK, S., GELATT, C. D., and VECCHI, M. P. (1983). Optimization by simulated annealing. *Science*, 220:671--680.

[12] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. 1992 *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, second edition, 1992.

[13] STORN, R. Differential Evolution Research – Trends and Open Questions

[14] GOLDBERG 1989 Genetic algorithms in search, optimisation and machine learning. Addison-Wesley, Reading, MA

[15] RUDOLPH 1996 Convergence of evolutionary algorithms in general search spaces. In: Proceedings of the third IEEE conference on evolutionary computation, IEEE Press, New York, pp 50-54

[16] JANEZ B, Saˇso Greiner, Borko Boˇskovic´, Marjan Mernik,Viljem ˇ Zumer, Self-Adapting Control Parameters in Differential Evolution: A Comparative Study on Numerical Benchmark Problems

[17] HELSGAUN, K. 2000. "An effective implementation of the Lin–Kernighan travelling salesman heuristic," European Journal of Operational Research, vol. 126, no. 1, pp. 106-130.

[18] MLADENOVIC N. 1995, "A variable neighbourhood algorithm – a new meta-heuristic for combinatorial optimization," in Abstracts of Papers at Optimization Days, Montreal, Canada, vol. 112.

[19] MLADENOVIC N, HANSEN, P, 1997: "Variable neighbourhood search," Computers and Operations research, vol. 24, no. 11, pp. 1097-1100.

Appendix A

Appendix A 1 Test-bed of 90 scenarios

Scenario	c	p	L_0	λ_1	λ_2	λ_3
1	1	9	1	0.01	0.01	0.01
2	1	9	1	0.01	0.01	0.1
3	1	9	1	0.01	0.01	1
4	1	9	1	0.01	0.1	0.1
5	1	9	1	0.1	0.1	0.1
6	1	9	1	1	0.1	0.1
7	1	9	1	0.01	0.1	1
8	1	9	1	1	1	1
9	1	9	1	1	1	0.1
10	1	9	1	1	1	0.01
11	1	9	5	0.01	0.01	0.01
12	1	9	5	0.01	0.01	0.1
13	1	9	5	0.01	0.01	1
14	1	9	5	0.01	0.1	0.1
15	1	9	5	0.1	0.1	0.1
16	1	9	5	1	0.1	0.1
17	1	9	5	0.01	0.1	1
18	1	9	5	1	1	1
19	1	9	5	1	1	0.1
20	1	9	5	1	1	0.01
21	1	9	10	0.01	0.01	0.01
22	1	9	10	0.01	0.01	0.1
23	1	9	10	0.01	0.01	1
24	1	9	10	0.01	0.1	0.1
25	1	9	10	0.1	0.1	0.1
26	1	9	10	1	0.1	0.1
27	1	9	10	0.01	0.1	1
28	1	9	10	1	1	1
29	1	9	10	1	1	0.1
30	1	9	10	1	1	0.01
31	1	999	1	0.01	0.01	0.01
32	1	999	1	0.01	0.01	0.1
33	1	999	1	0.01	0.01	1
34	1	999	1	0.01	0.1	0.1
35	1	999	1	0.1	0.1	0.1
36	1	999	1	1	0.1	0.1
37	1	999	1	0.01	0.1	1
38	1	999	1	1	1	1
39	1	999	1	1	1	0.1
40	1	999	1	1	1	0.01
41	1	999	5	0.01	0.01	0.01
42	1	999	5	0.01	0.01	0.1
43	1	999	5	0.01	0.01	1
44	1	999	5	0.01	0.1	0.1
45	1	999	5	0.1	0.1	0.1
46	1	999	5	1	0.1	0.1
47	1	999	5	0.01	0.1	1
48	1	999	5	1	1	1
49	1	999	5	1	1	0.1
50	1	999	5	1	1	0.01
51	1	999	10	0.01	0.01	0.01
52	1	999	10	0.01	0.01	0.1

53	1	999	10	0.01	0.01	1
54	1	999	10	0.01	0.1	0.1
55	1	999	10	0.1	0.1	0.1
56	1	999	10	1	0.1	0.1
57	1	999	10	0.01	0.1	1
58	1	999	10	1	1	1
59	1	999	10	1	1	0.1
60	1	999	10	1	1	0.01
61	1	99999	1	0.01	0.01	0.01
62	1	99999	1	0.01	0.01	0.1
63	1	99999	1	0.01	0.01	1
64	1	99999	1	0.01	0.1	0.1
65	1	99999	1	0.1	0.1	0.1
66	1	99999	1	1	0.1	0.1
67	1	99999	1	0.01	0.1	1
68	1	99999	1	1	1	1
69	1	99999	1	1	1	0.1
70	1	99999	1	1	1	0.01
71	1	99999	5	0.01	0.01	0.01
72	1	99999	5	0.01	0.01	0.1
73	1	99999	5	0.01	0.01	1
74	1	99999	5	0.01	0.1	0.1
75	1	99999	5	0.1	0.1	0.1
76	1	99999	5	1	0.1	0.1
77	1	99999	5	0.01	0.1	1
78	1	99999	5	1	1	1
79	1	99999	5	1	1	0.1
80	1	99999	5	1	1	0.01
81	1	99999	10	0.01	0.01	0.01
82	1	99999	10	0.01	0.01	0.1
83	1	99999	10	0.01	0.01	1
84	1	99999	10	0.01	0.1	0.1
85	1	99999	10	0.1	0.1	0.1
86	1	99999	10	1	0.1	0.1
87	1	99999	10	0.01	0.1	1
88	1	99999	10	1	1	1
89	1	99999	10	1	1	0.1
90	1	99999	10	1	1	0.01

Scenarios	c	p	L_0	λi	No of retailers
1	1	9	10	$\lambda i = \Re[0,1]$	3
2	1	9	10	$\lambda i = \Re[0,1]$	6
3	1	9	10	$\lambda i = \Re[0,1]$	12
4	1	9	10	$\lambda i = \Re[0,1]$	24
5	1	9	10	$\lambda i = \Re[0,1]$	36
6	1	9	10	$\lambda i = \Re[0,1]$	48
7	1	9	10	$\lambda i = \Re[0,1]$	60
8	1	999	10	$\lambda i = \Re[0,1]$	3
9	1	999	10	$\lambda i = \Re[0,1]$	6
10	1	999	10	$\lambda i = \Re[0,1]$	12
11	1	999	10	$\lambda i = \Re[0,1]$	24
12	1	999	10	$\lambda i = \Re[0,1]$	36
13	1	999	10	$\lambda i = \Re[0,1]$	48

14	1	999	10	$\lambda i = \Re[0,1]$	60
15	1	99999	10	$\lambda i = \Re[0,1]$	3
16	1	99999	10	$\lambda i = \Re[0,1]$	6
17	1	99999	10	$\lambda i = \Re[0,1]$	12
18	1	99999	10	$\lambda i = \Re[0,1]$	24
19	1	99999	10	$\lambda i = \Re[0,1]$	36
20	1	99999	10	$\lambda i = \Re[0,1]$	48
21	1	99999	10	$\lambda i = \Re[0,1]$	60

Note: The arrival rates are real values between 0 and 1...